KDB kernel debugger and kdb command
KDB kernel debugger and kdb command
Fourth Edition (July 2006)

A reader’s comment form is provided at the back of this publication. If the form has been removed, address comments to Information Development, Department 04XA-905-6C006, 11501 Burnet Road, Austin, Texas 78758-3493. To send comments electronically, use this commercial Internet address: aix6kpub@austin.ibm.com. Any information that you supply may be used without incurring any obligation to you.

This edition applies to AIX 5L Version 5.3 and to all subsequent releases of this product until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1997, 2006. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About This Book</td>
<td>ix</td>
</tr>
<tr>
<td>How to Use This Book</td>
<td>ix</td>
</tr>
<tr>
<td>Highlighting</td>
<td>ix</td>
</tr>
<tr>
<td>Case-Sensitivity in AIX</td>
<td>x</td>
</tr>
<tr>
<td>ISO 9000</td>
<td>x</td>
</tr>
<tr>
<td>Related Publications</td>
<td>x</td>
</tr>
<tr>
<td>Chapter 1. KDB kernel debugger and kdb command</td>
<td>1</td>
</tr>
<tr>
<td>KDB kernel debugger</td>
<td>1</td>
</tr>
<tr>
<td>The kdb command</td>
<td>5</td>
</tr>
<tr>
<td>Chapter 2. The debugger prompt</td>
<td>7</td>
</tr>
<tr>
<td>Online help</td>
<td>7</td>
</tr>
<tr>
<td>Registers</td>
<td>8</td>
</tr>
<tr>
<td>Expressions</td>
<td>10</td>
</tr>
<tr>
<td>User-defined variables</td>
<td>10</td>
</tr>
<tr>
<td>Command line editing</td>
<td>11</td>
</tr>
<tr>
<td>Multiprocessor systems</td>
<td>12</td>
</tr>
<tr>
<td>Chapter 3. Viewing and modifying global data</td>
<td>13</td>
</tr>
<tr>
<td>Method 1: Using the symbol name</td>
<td>13</td>
</tr>
<tr>
<td>Method 2: Using the TOC and map file</td>
<td>14</td>
</tr>
<tr>
<td>Method 3: Using the map file</td>
<td>15</td>
</tr>
<tr>
<td>Chapter 4. Viewing stack traces</td>
<td>17</td>
</tr>
<tr>
<td>Stack frame format</td>
<td>17</td>
</tr>
<tr>
<td>Verbose stack output</td>
<td>19</td>
</tr>
<tr>
<td>Chapter 5. Setting breakpoints</td>
<td>21</td>
</tr>
<tr>
<td>Method 1: Using the lke subcommand</td>
<td>21</td>
</tr>
<tr>
<td>Method 2: Using the nm subcommand</td>
<td>22</td>
</tr>
<tr>
<td>Method 3: Using the kmid pointer</td>
<td>23</td>
</tr>
<tr>
<td>Method 4: Using the devsw subcommand</td>
<td>23</td>
</tr>
<tr>
<td>Chapter 6. Using KDB kernel debugger to perform a trace</td>
<td>27</td>
</tr>
<tr>
<td>Chapter 7. Subcommand lists</td>
<td>29</td>
</tr>
<tr>
<td>Alphabetic list</td>
<td>29</td>
</tr>
<tr>
<td>Task category list</td>
<td>39</td>
</tr>
<tr>
<td>Chapter 8. End user subcommands</td>
<td>41</td>
</tr>
<tr>
<td>h subcommand</td>
<td>42</td>
</tr>
<tr>
<td>set subcommand</td>
<td>44</td>
</tr>
<tr>
<td>dbgopt subcommand</td>
<td>47</td>
</tr>
<tr>
<td>varset subcommand</td>
<td>48</td>
</tr>
<tr>
<td>varlist subcommand</td>
<td>49</td>
</tr>
<tr>
<td>varrm subcommand</td>
<td>50</td>
</tr>
<tr>
<td>his subcommand</td>
<td>51</td>
</tr>
<tr>
<td>debug subcommand</td>
<td>52</td>
</tr>
<tr>
<td>l subcommand</td>
<td>54</td>
</tr>
<tr>
<td>Chapter 9. Leaving kdb subcommands</td>
<td>55</td>
</tr>
<tr>
<td>e subcommand</td>
<td>56</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1997, 2006
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Subcommand(s)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Changing context subcommands</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>reboot subcommand</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>halt subcommand</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>sw subcommand</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>cpu subcommand</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>ctx subcommand</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>runcpu subcommand</td>
<td>67</td>
</tr>
<tr>
<td>11</td>
<td>Calculator and converter subcommands</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>hcal and dcal subcommands</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>conv subcommand</td>
<td>71</td>
</tr>
<tr>
<td>12</td>
<td>CPU start and stop subcommands</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>start and stop subcommands</td>
<td>74</td>
</tr>
<tr>
<td>13</td>
<td>Basic display subcommands</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>f subcommand</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>status subcommand</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>stat subcommand</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>pr subcommand</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>symptom subcommand</td>
<td>87</td>
</tr>
<tr>
<td>14</td>
<td>Memory register display and decode subcommands</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>d, dw, dd, dp, dpw, and dpd subcommands</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>dc and dpc subcommands</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>di subcommand</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>dr subcommand</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>ddvb, ddvh, ddvd, ddpv, ddpb, ddpv, ddpw, and ddpd subcommands</td>
<td>98</td>
</tr>
<tr>
<td>15</td>
<td>Memory search and extract subcommands</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>find and findp subcommands</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>ext and exp subcommands</td>
<td>104</td>
</tr>
<tr>
<td>16</td>
<td>Memory modification subcommands</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>m, mw, md, mp, mpw, and mpd subcommands</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>st, stc, and sth subcommands</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>mdvb, mdvh, mdvd, mdvb, mdpb, mdph, mdpw, mdpd subcommands</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>mr subcommand</td>
<td>113</td>
</tr>
<tr>
<td>17</td>
<td>Breakpoint and steps subcommands</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>b subcommand</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>lb subcommand</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>c, lc, and ca subcommands</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>r and gt subcommands</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>n, s, S, and B subcommand</td>
<td>124</td>
</tr>
<tr>
<td>18</td>
<td>Debugger trace points subcommands</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>bt subcommand</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>test subcommand</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>cat and ct subcommands</td>
<td>131</td>
</tr>
<tr>
<td>19</td>
<td>Watch DABR subcommands</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>wr, ww, wrw, cw, lwr, lww, lww, and lcw subcommands</td>
<td>134</td>
</tr>
<tr>
<td>20</td>
<td>Branch target subcommands</td>
<td>137</td>
</tr>
</tbody>
</table>
btac, cbtac, lbtac, lcbtac subcommands .................................................. 138

Chapter 21. Namelist and symbols subcommands ....................................... 141
  nm and ts subcommands ........................................................................ 142
  ns subcommand .................................................................................... 143
  which subcommand ............................................................................... 144

Chapter 22. PCI configuration space and I/O debugging subcommands ............... 145
  dpcib, dpcih, and dpciw subcommand .................................................. 146
  mpcib, mpcih, and mpciw subcommands ............................................. 147
  buserr subcommand ............................................................................ 149
  businfo subcommand .......................................................................... 152

Chapter 23. Display kernel data structures subcommands ............................... 155
  var subcommand .................................................................................. 156
  drvars subcommand ............................................................................ 158
  ipl subcommand .................................................................................. 159
  devsw subcommand ............................................................................ 161
  intr subcommand ................................................................................ 162

Chapter 24. Display VMM subcommands ..................................................... 165
  ames subcommand ............................................................................... 167
  apt subcommand .................................................................................. 169
  frameset subcommand ....................................................................... 171
  free subcommand ............................................................................... 172
  freelist subcommand ......................................................................... 173
  ipc subcommand ................................................................................. 174
  rtipc subcommand .............................................................................. 176
  rtipcd subcommand .......................................................................... 178
  lka subcommand ................................................................................ 180
  lkh subcommand ................................................................................ 181
  lkw subcommand ............................................................................... 182
  mempool subcommand ....................................................................... 184
  pdt subcommand ................................................................................ 186
  pfhd data subcommand ..................................................................... 188
  pft subcommand ................................................................................. 190
  sw what subcommand ......................................................................... 194
  pv t subcommand ............................................................................... 196
  pta subcommand ............................................................................... 198
  pte subcommand ............................................................................... 199
  rmap subcommand ............................................................................. 201
  rv sid subcommand ............................................................................ 203
  scb subcommand ............................................................................... 204
  segst64 subcommand ....................................................................... 207
  sr64 subcommand .............................................................................. 208
  ksp subcommand ............................................................................... 211
  ste subcommand ................................................................................ 213
  vmbufst subcommand ....................................................................... 215
  vmaddr subcommand ......................................................................... 217
  vmdmap subcommand ....................................................................... 218
  vmint subcommand ............................................................................ 220
  vmker subcommand ........................................................................... 223
  vlocks subcommand .......................................................................... 225
  vmsg subcommand ............................................................................ 227
  vmpool subcommand ......................................................................... 228
  vmstat subcommand .......................................................................... 230
vmthrpgio subcommand .................................................. 232
vmwait subcommand ...................................................... 233
vrld subcommand .......................................................... 234
vsidd subcommand .......................................................... 235
vsidm subcommand ........................................................ 236
zproc subcommand ........................................................ 237
drlist subcommand ......................................................... 238
lrustate subcommand ..................................................... 239

Chapter 25. Address translation subcommands .......................... 241
tr and tv subcommands ................................................... 242
slb subcommand ............................................................ 243
mslb subcommand ........................................................... 245
dbat subcommand ............................................................ 246
ibat subcommand ............................................................ 247
mdbat subcommand .......................................................... 248
mibat subcommand .......................................................... 249

Chapter 26. Loader subcommands ........................................... 251
lke, stlb, and rmst subcommand ...................................... 252
lle subcommand ............................................................. 256
exp subcommand ............................................................. 259

Chapter 27. Display context information subcommands .................. 261
pnda subcommand .......................................................... 262
ppda subcommand .......................................................... 264
mst subcommand ............................................................ 266
lastbackt subcommand ................................................... 268
ppid subcommand ........................................................... 270
ptid subcommand ........................................................... 271
ttid subcommand ............................................................. 273
tpid subcommand ............................................................ 275
rq subcommand ............................................................... 276
rqi subcommand ............................................................. 278
lg subcommand ............................................................... 280
cr subcommand ............................................................... 281
svmon subcommand ........................................................ 282
meml subcommand .......................................................... 284
cred subcommand ........................................................... 285

Chapter 28. Display storage subsystem information subcommands .......... 287
pbuf subcommand ........................................................... 288
volgrp subcommand ......................................................... 289
pvol subcommand ............................................................ 291
lvol subcommand ............................................................ 292
scd subcommand ............................................................. 293

Chapter 29. Display memory allocation information subcommands ........... 297
heap subcommand ............................................................. 298
xmalloc subcommand ......................................................... 303
kmbucket subcommand .................................................... 306
kmstats subcommand ....................................................... 308

Chapter 30. Display general and Journal File System (JFS) information subcommands .... 311
dnlc subcommand ............................................................ 312
hdnlc subcommand ........................................................... 314
Chapter 31. Display Enhanced Journaled File System information subcommands .......................... 349
j2 subcommand .................................................................................................................. 350
i2 subcommand .................................................................................................................. 351
tree subcommand .............................................................................................................. 353
dtree subcommand ............................................................................................................. 354
xtree subcommand ............................................................................................................. 355
pgobj subcommand ............................................................................................................ 356
pgbuf subcommand ............................................................................................................ 357
txblock subcommand ......................................................................................................... 358
txblocki subcommand ........................................................................................................ 359
tlock subcommand ............................................................................................................. 360
bmblock subcommand ........................................................................................................ 361
jfs2node subcommand ....................................................................................................... 362
j2logbuf subcommand ........................................................................................................ 364
j2logx subcommand ........................................................................................................... 365
j2log subcommand .............................................................................................................. 366
pile subcommand ............................................................................................................... 367
slab subcommand ............................................................................................................... 369

Chapter 32. Display NFS information subcommands ............................................................. 371
cupboard subcommand ........................................................................................................ 372
sockpint subcommand ........................................................................................................ 374
sockcup subcommand ........................................................................................................ 375
svcxprt subcommand ......................................................................................................... 376

Chapter 33. Time subcommands. .......................................................................................... 377
time subcommand .............................................................................................................. 378
trb subcommand ................................................................................................................ 380

Chapter 34. System trace, dump and error log subcommands ................................................ 383
trace subcommand ............................................................................................................. 384
trcstart subcommand ......................................................................................................... 386
trcstop subcommand .......................................................................................................... 387
mtrace subcommand ......................................................................................................... 388
cdt subcommand .............................................................................................................. 390
About This Book

This book describes how to use the KDB kernel debugger and the kdb command to debug an operating system image. It describes how to examine a stopped kernel in the KDB kernel debugger, as well as how to examine a system dump file using the kdb command. It provides a reference for the commands used to debug the kernel, device drivers, and other kernel extensions for AIX 5L™. Topics include setting breakpoints within the kernel or in kernel extensions, displaying and modifying data structures and instructions, altering system registers, and performing traces. Specific information (for example, syntax and description) is given for each subcommand.

How to Use This Book

Read the beginning chapters of this book to learn about the KDB kernel debugger and the kdb command.

You can use the alphabetical list to locate a specific subcommand. The subcommand list includes the aliases for each subcommand, a short description of its use, information on when the subcommand can be used and the category to which the subcommand belongs. Subcommands and their aliases are also included in the index.

Reading Syntax Statements

Syntax statements are a way to represent subcommand syntax and consist of symbols such as brackets ([ ]), braces ({ }), and vertical bars (|). The following is a sample syntax statement:

```
examplesubcommand [ a | b ] [ -x value ] [ -y { address | symbol } ] [ -z ] filename ...
```

The following conventions are used in the command syntax statements:

- Items that must be entered literally on the command line are in bold. These items include the command name, flags, and literal characters.
- Items representing variables that must be replaced by a name are in italics. These items include parameters that follow flags and parameters that the command reads, such as Files and Directories.
- Parameters enclosed in brackets are optional.
- Parameters enclosed in braces are required.
- Parameters not enclosed in either brackets or braces are required.
- A vertical bar signifies that you choose only one parameter. For example, [ a | b ] indicates that you can choose a, b, or nothing. Similarly, { a | b } indicates that you must choose either a or b.
- Ellipses ( ... ) signify the parameter can be repeated on the command line.
- The dash ( - ) represents standard input.

Highlighting

The following highlighting conventions are used in this book:

**Bold**

Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.

*Italics*

Identifies parameters whose actual names or values are to be supplied by the user.

**Monospace**

Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.
Case-Sensitivity in AIX

Everything in the AIX® operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the ls command to list files. If you type LS, the system responds that the command is "not found." Likewise, FILEA, FiLea, and filea are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related Publications

The following books contain information about or related to debugging programs:

- AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts
Chapter 1. KDB kernel debugger and kdb command

This document describes the KDB kernel debugger and kdb command. The KDB kernel debugger and the kdb command are the primary tools a developer uses for debugging device drivers, kernel extensions, and the kernel itself. Although they appear similar to the user, the KDB kernel debugger and the kdb command are two separate tools:

KDB kernel debugger
The KDB kernel debugger is integrated into the kernel and allows full control of the system while a debugging session is in progress. The KDB kernel debugger allows for traditional debugging tasks such as setting breakpoints and single-stepping through code.

kdb command
This command is implemented as an ordinary user-space program and is typically used for post-mortem analysis of a previously-crashed system by using a system dump file. The kdb command includes subcommands specific to the manipulation of system dumps.

Both the KDB kernel debugger and kdb command allow the developer to display various structures normally found in the kernel’s memory space. Both do the following:

- Provide numerous subcommands to decode various data structures found throughout the kernel.
- Print the data structures in a user-friendly format.
- Perform debugging at the machine instruction level. Although this is less convenient than source level debugging, it allows the KDB kernel debugger and the kdb command to be used in the field where access to source code might not be possible.
- Process the debugging information found in XCOFF objects. This allows the use of symbolic names for functions and global variables.

The following sections describe more about the KDB kernel debugger and kdb command:

- "KDB kernel debugger"
- "The kdb command" on page 5

The following sections outline how to invoke the KDB kernel debugger and kdb command:

- "Invoking the KDB kernel debugger" on page 2
- "Invoking the kdb command" on page 6

KDB kernel debugger

Although it must be manually enabled by the user prior to use, the KDB kernel debugger is statically compiled into the AIX kernel and is always loaded. After it is enabled, the KDB kernel debugger can be manually invoked by the user or automatically invoked by the system in response to some condition (for example, an unhandled exception in the kernel code). For more information, see "Invoking the KDB kernel debugger" on page 2.

KDB kernel debugger is always loaded into a special region of pinned memory where the effective address space equals the real address space. The KDB kernel debugger runs with memory translation turned off. This allows it to function even if the VMM subsystem is not yet initialized or the critical VMM structures are corrupted. However, the KDB kernel debugger can perform the same address translations normally performed by the processor. This allows the user to view data by effective addresses when the processor has its memory translation turned off.

When the KDB kernel debugger is invoked by a condition, it is the only running program. All other processes are stopped and processor interrupts are disabled. One of the processors is designated as the debug processor and that processor runs the KDB kernel debugger. This is usually the processor on which an unusual activity occurred (for example, an unhandled exception).
If the KDB kernel debugger is invoked manually by the user, the debug processor is arbitrarily chosen. The KDB kernel debugger stops all other processors in the system by sending an interprocessor interrupt (IPI) to each processor. If any of these processors cannot be stopped, the KDB kernel debugger prints a warning message. For example, if a processor is spinning on a lock with interrupts disabled, it cannot process the IPI sent by the KDB kernel debugger.

The KDB kernel debugger is mostly self-contained and does not rely on other kernel components such as the network and video drivers. The KDB kernel debugger runs with its own Machine State Save Area (mst) and a special stack. This requires that some kernel code be duplicated within KDB kernel debugger. Duplication allows the developer to debug from almost anywhere within the kernel code. Unless the KDB kernel debugger is entered through a system halt, processors resume normal operation and interrupts are re-enabled when the developer exits the KDB kernel debugger.

When it is invoked, the KDB kernel debugger takes control of either the virtual terminal (vterm) on a logical partitioning system, or a physical RS232 serial port on a non-logical partitioning system. This requires a Hardware Management Console (HMC) to access the vterm or another system connected to the serial port on the system being debugged. The KDB kernel debugger requires the connection in order to send messages to the developer.

The complete list of subcommands available for the KDB kernel debugger and kdb command are included in Chapter 7, “Subcommand lists,” on page 29.

**Invoking the KDB kernel debugger**

This topic describes how to load and start the KDB kernel debugger, and what you need to know about terminal use. For information on how to invoke the kdb command, see “Invoking the kdb command” on page 6.

**Loading and starting the KDB kernel debugger in AIX 5.1 and subsequent releases**

For AIX 5.1 and subsequent releases, the KDB kernel debugger is the standard kernel debugger and is included in the unix_up and unix_mp kernels, which are in the /usr/lib/boot file.

The KDB kernel debugger must be loaded at boot time. This requires that a boot image be created with the debugger enabled. To enable the KDB kernel debugger, use either the -I or -D options of the bosboot command.

Examples of bosboot commands are as follows:

- To disable the KDB kernel debugger, use the following command:
  
  ```
  bosboot -a -d /dev/ipldevice
  ```

- To enable the KDB kernel debugger, but not invoke it during system initialization, use the following command:
  
  ```
  bosboot -a -d /dev/ipldevice -D
  ```

- To enable the KDB kernel debugger, and invoke it during system initialization, use the following command:
  
  ```
  bosboot -a -d /dev/ipldevice -I
  ```

**Notes:**

1. **bosboot** commands build boot images using the KDB kernel debugger. The boot image is not used until the machine is restarted.
2. External interrupts are disabled while the KDB kernel debugger is active.
3. If invoked during system initialization, the **g** subcommand must be issued to continue the initialization process.

For more information on the **bosboot** command, see [AIX 5L Version 5.3 Commands Reference, Volume 1](#).
Loading and starting the KDB kernel debugger in AIX 4.3.3
The KDB kernel debugger must be loaded at boot time. This requires that a boot image be created with the debugger enabled. To enable the KDB kernel debugger, the **bosboot** command must be invoked with a KDB kernel specified and options set to enable the KDB kernel debugger. KDB kernels are shipped as /usr/lib/boot/unix_kdb for uni-processor (UP) systems and /usr/lib/boot/unix_mp_kdb for Multi-processor (MP) systems. The specific kernel used to create the boot image can be specified using the -k option of the **bosboot** command. The KDB kernel debugger must also be enabled using either the -I or -D options of the **bosboot** command.

Examples of **bosboot** commands for a UP system are as follows:

- To disable the KDB kernel debugger, use the following command:
  
  ```bash
  bosboot -a -d /dev/ipldevice -k /usr/lib/boot/unix_kdb
  ```

- To enable the KDB kernel debugger, but not invoke it during system initialization, use the following command:
  
  ```bash
  bosboot -a -d /dev/ipldevice -D -k /usr/lib/boot/unix_kdb
  ```

- To enable the KDB kernel debugger, and invoke it during system initialization, use the following command:
  
  ```bash
  bosboot -a -d /dev/ipldevice -I -k /usr/lib/boot/unix_kdb
  ```

**Notes:**
1. For an MP system, the /usr/lib/boot/unix_mp_kdb file is used instead of the /usr/lib/boot/unix_kdb file.
2. The **bosboot** commands build boot images using the KDB kernel debugger. The boot image is not used until the machine is restarted.
3. External interrupts are disabled while the KDB kernel debugger is active.
4. If invoked during system initialization, the **g** subcommand must be issued to continue the initialization process.

For more information about the **bosboot** command, see [AIX 5L Version 5.3 Commands Reference, Volume 1](https://publib.boulder.ibm.com/infocenter/systems/v53r1/topic/com.ibm.aix.doc/COMM/commn.html).

The /usr/lib/boot/unix and /unix links are not changed by the **bosboot** command. However, these links are used by user commands such as **sar** and others to read symbol information for the kernel. If these commands are to be used with a KDB boot image /unix and /usr/lib/boot/unix must point to the kernel specified for the **bosboot** command. This can be done by removing and recreating the links. This must be done as the root user. For the previous **bosboot** command examples, typing the following would set up the links correctly:

1. Type
   
   ```bash
   rm /unix
   ```
   and press Enter.

2. Type
   
   ```bash
   ln -s /usr/lib/boot/unix_kdb /unix
   ```
   and press Enter.

3. Type
   
   ```bash
   rm /usr/lib/boot/unix
   ```
   and press Enter.

4. Type
   
   ```bash
   ln -s /usr/lib/boot/unix_kdb /usr/lib/boot/unix
   ```
   and press Enter.
Similarly, if you chose to stop using a KDB Kernel, the links for `/unix` and `/usr/lib/boot/unix` should be modified to point to the kernel specified to the `bosboot` command.

**Note:** `/unix` is the default kernel used by the `bosboot` command. If this link is changed to point to a KDB kernel, after `bosboot` commands that do not have a kernel specified are run, the commands use the KDB kernel.

**Entering the KDB kernel debugger**
Enter the KDB kernel debugger using one of the following procedures:

- On a tty keyboard, press the Ctrl+4 key sequence for IBM® 3151 terminals or the Ctrl+\ key sequence for BQ 303, BQ 310C, and WYSE 50 terminals.
- On other keyboards, press the Ctrl+Alt+Numpad4 key sequence.
- Set a breakpoint using one of the Chapter 17, “Breakpoint and steps subcommands,” on page 115.
- Call the `brkpoint` subroutine from the C code. The syntax for calling this subroutine is the following:
  
  ```c
  brkpoint();
  ```

**Note:** The system enters the debugger if a system halt is caused by a fatal system error. In such a case, the system creates a log entry in the system log and if the KDB kernel debugger is available, it is called. A system dump might be generated when you exit from the debugger.

If the kernel debug program is not available when you type in a key sequence, you must load the kernel debug program.

For more information about loading the kernel debug program, see “Loading and starting the KDB kernel debugger in AIX 4.3.3” on page 3 or “Loading and starting the KDB kernel debugger in AIX 5.1 and subsequent releases” on page 2.

You can use the `kdb` command with the `dw` subcommand to determine whether the KDB kernel debugger is available by typing the following:

```
# kdb
(0)> dw kdb_avail
(0)> dw kdb_wanted
```

**Note:** If either of the previous `dw` subcommands returns a 0, the KDB kernel debugger is not available.

After the KDB kernel debugger is invoked, the subcommands detailed in Chapter 7, “Subcommand lists,” on page 29 are available.

**Using a terminal with the KDB kernel debugger**

**Note:** If you are using the Hardware Management Console, KDB kernel debugger can be accessed using a virtual terminal. For more information, see the Hardware Management Console Installation and Operations Guide (SA38 – 0590).

The KDB kernel debugger opens an asynchronous ASCII terminal when it is first started, and subsequently upon being started due to a system halt. Native serial ports are checked sequentially, starting with port 0 (zero). Each port is configured at 9600 bps, 8 bits, and no parity. If carrier detect is asserted within 1/10 of a second, the port is used. Otherwise, the next available native port is checked. This process continues until a port is opened or until every native port available on the machine is checked. If no native serial port is opened successfully, the result is unpredictable.

The KDB kernel debugger only supports display to an ASCII terminal connected to a native serial port. Displays connected to graphics adapters are not supported. The KDB kernel debugger uses its own device driver for handling the display terminal. It is possible to connect a serial line between two machines and...
define the serial line port as the port for the console. In that case, the `cu` command can be used to connect to the target machine and run the KDB kernel debugger.

**Note:** If a serial device, other than a terminal connected to a native serial port, is selected by the kernel debugger, the system might appear to hang.

## The kdb command

The `kdb` command can be used for analyzing the following:

- **A running system.**
  
  When used to analyze a running system, the `kdb` command opens the `/dev/pmem` special file, which allows direct access to the system's physical memory and bypasses the normal address translation mechanism of the processor. The `kdb` command performs its own address translation internally using the same algorithms as the KDB kernel debugger. This allows the user to view data by effective address.

  **Note:** Only the root user can use the `kdb` command to analyze a running system.

- **A system dump file produced by a previously crashed-system.**
  
  When a system crashes, the system dump image is created with memory translation turned on. As a result, any physical memory not mapped to the effective address space at the time of the dump cannot be included in the dump file. Only the memory belonging to the process that was running on the processor that created the dump image can be included in the dump file. Because all addresses within the system dump are already effective addresses, the `kdb` command does not perform its internal address translation.

  A system dump contains certain critical data structures. A system dump does not contain the entire effective address space. The `kdb` command might not be able to view certain memory regions. If someone attempts to access a memory address not included in the dump, the `kdb` command prints a warning message.

  When analyzing a system dump, it is imperative that the `kdb` command uses the same version of the UNIX file that was running at the time of the dump. To check the time stamps of dump and UNIX files, use the following commands:

  ```bash
  $ what unix | grep _kdb_buildinfo
  _kdb_buildinfo unix_64 Mar 14 2005 10:24:29 (This is the return that users will get.)
  $ what dump | grep _kdb_buildinfo
  _kdb_buildinfo unix_64 Mar 14 2005 10:24:29
  ```

  The time stamps of both files must be identical. It is also possible to check the time stamp of the `kdb` command by running the following commands:

  ```bash
  $ what /usr/sbin/kdb_64 | grep _kdb_buildinfo
  _kdb_buildinfo unix_64 Mar 4 2005 14:45:20
  $ what /usr/sbin/kdb_mp | grep _kdb_buildinfo
  _kdb_buildinfo unix_mp Mar 4 2005 14:31:53
  ```

  This time stamp will typically be older than that of the dump and UNIX files. Usually, the `kdb` command can read the dump in this condition. However, if the version difference between the `kdb` command and the UNIX file is too large, `kdb` might be unable to read the dump. In this case, use a version of `kdb` that is closer to the UNIX version that is used.

  **Note:** The `cdt` subcommand or the `-v` command-line option can be used to determine exactly which regions of the effective address space are included in the system image. For more information about the `CDT` subcommand, see the `cdt subcommand` on page 390. For more information about the `-v` command line option, see Appendix A, "kdb Command," on page 441.

The `kdb` command contains a subset of the subcommands found in the KDB kernel debugger. Subcommands for setting breakpoints and single-stepping through code are not available in the `kdb` command. Because the `kdb` command is implemented as an ordinary user-space program, it has no control over the processors in a system. Similarly, any subcommands that directly access hardware (for
example, the PCI subcommands) are not available. When you work with a system dump, any subcommands that modify memory are not valid because the system dump is merely a snapshot of the real memory in a system.

The complete list of subcommands available for the KDB kernel debugger and kdb command are included in Chapter 7, “Subcommand lists,” on page 29.

**Invoking the kdb command**

This topic describes how to configure a processor for system dumps, obtain and verify a system dump, and run the kdb command. To analyze a running system, the kdb command is simply invoked from the UNIX® shell prompt without any command line arguments.

**Note:** Because the kdb command makes use of the /dev/pmem special file when analyzing a running system, only the root user can invoke the command in this manner.

A side effect of analyzing the running system with the kdb command is that the currently running process as displayed with the p * subcommand, often appears to be the kdb command itself. This occurs because the kdb command can only read the /dev/pmem special file when it is the current process on one of the processors in the system.

When you are analyzing a system dump file, the kdb command must be started with command line arguments that specify the location of the dump files and the kernel files as shown in the following example:

```
# kdb /var/adm/ras/vmcore.0 /unix
```

The kernel file is used by the kdb command to resolve symbol names from the dump file. It is imperative that the kernel file specified on the command line is the kernel file that was running at the time the system dump was created.

For more information about creating system dumps, see System Dump Facility in AIX® 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts.

For more information about invoking the KDB kernel debugger, see “Invoking the KDB kernel debugger” on page 2.
Chapter 2. The debugger prompt

All work in the KDB kernel debugger and the kdb command is performed at the debugger prompt. On a uniprocessor system, the KDB kernel debugger prompt is KDB(0)> and the kdb command prompt is (0)>.

When you are debugging a multiprocessor system, the number enclosed in parentheses indicates the processor that is being debugged. Many subcommands, such as those that display or modify registers, apply only to the current processor.

As shown in the following example, the cpu subcommand can be used to change the current processor:

(0)> dr r1
r1 : 2FF3B338 2FF3B338
(0)> cpu 1
(1)> dr r1
r1 : 2FF3AA20 2FF3AA20
(1)>

Many subcommands can produce a large amount of output. To keep the output from scrolling off the screen, the debugger implements a pager which displays a more ("^C to quit") ? prompt after each full screen of data. When you see the prompt, you can do one of the following:

- Press the space bar to view the next line of output.
- Press the Enter key to view the next page of output.
- Press Ctrl+C to abort the current subcommand and return to the main debugger prompt.

The pager is controlled with the set subcommand using the screen_size and scroll options. For more information, see the "set subcommand" on page 44.

Online help

The help subcommand can be typed at any time to display a list of all available subcommands and a one-line description of each of the subcommands. Many subcommands also allow a -? parameter that displays a more detailed description of that subcommand. For example, to see a list of display context subcommands, type the following at the command prompt:

help display context

The following results are displayed:

<table>
<thead>
<tr>
<th>CMD</th>
<th>ALIAS</th>
<th>ALIAS</th>
<th>FUNCTION</th>
<th>ARG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*** display context information ***</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pnda</td>
<td>Display pnd area</td>
<td>[*][-a][cpnb/symb/eaddr]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppda</td>
<td>Display ppd area</td>
<td>[*]/cpnb/symb/eaddr]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mst</td>
<td>Display mst area</td>
<td>[slot] [[-a] symb/eaddr]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lastbackt</td>
<td>Display lastbackt</td>
<td>cpu number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>proc</td>
<td>Display proc table</td>
<td>[*/slot/symb/eaddr]</td>
<td></td>
</tr>
<tr>
<td>th</td>
<td>thread</td>
<td>Display thread table</td>
<td>[*/slot/symb/eaddr/-w?]</td>
<td></td>
</tr>
<tr>
<td>ttid</td>
<td>th_ttid</td>
<td>Display thread tid</td>
<td>[tid]</td>
<td></td>
</tr>
<tr>
<td>tpid</td>
<td>th_pid</td>
<td>Display thread pid</td>
<td>[pid]</td>
<td></td>
</tr>
<tr>
<td>rq</td>
<td>runq</td>
<td>Display run queues</td>
<td>[bucket/symb/eaddr]</td>
<td></td>
</tr>
<tr>
<td>rq1</td>
<td>rqa</td>
<td>Display RQ Info</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sq</td>
<td>sleepq</td>
<td>Display sleep queues</td>
<td>[bucket/symb/eaddr]</td>
<td></td>
</tr>
<tr>
<td>lq</td>
<td>lockq</td>
<td>Display lock queues</td>
<td>[bucket/symb/eaddr]</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>user</td>
<td>Display u_area</td>
<td>[-?] [slot/symb/eaddr]</td>
<td></td>
</tr>
<tr>
<td>cr</td>
<td>crid</td>
<td>Display crid table</td>
<td>[*/slot/symb/eaddr]</td>
<td></td>
</tr>
<tr>
<td>chkfile</td>
<td>Display chkfile structure</td>
<td>eaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>svmn</td>
<td>Process based paging space and mem usage</td>
<td>[-?]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example, to see a list of parameters for the p subcommand and a brief description of what the parameter does, type the following at the command prompt:
The following results are displayed:

```
PROC USAGE:'p ?' print usage
PROC USAGE:'p' print current process
PROC USAGE:'p *' print process table
PROC USAGE: 'p -' print all processes in none/zombie state in long format
PROC USAGE: 'p <slot>' print process in <slot>
PROC USAGE: 'p <address>' print process at <address>
PROC USAGE: 'p <symbol>' print process matching <symbol>
PROC USAGE: 'p -s <proc state>' sort processes by state
PROC USAGE: 'p -n <substring>' sort processes by name
```

For an alphabetic list of the subcommands, see Chapter 7, “Subcommand lists,” on page 29. Because the -? parameter is available with most subcommands, this parameter is not included in the detailed subcommand descriptions found in this book.

### Registers

Register values can be referenced by the KDB kernel debugger and the `kdb` command. Register values can be used in subcommands by preceding the register name with an at sign (@). This character is also used to dereference addresses as described in “Expressions” on page 10. Registers that can be referenced include the following:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asr</td>
<td>Address space register</td>
</tr>
<tr>
<td>cr</td>
<td>Condition register</td>
</tr>
<tr>
<td>ctr</td>
<td>Count register</td>
</tr>
<tr>
<td>dar</td>
<td>Data address register</td>
</tr>
<tr>
<td>dec</td>
<td>Decrementer</td>
</tr>
<tr>
<td>dsisr</td>
<td>Data storage interrupt status register</td>
</tr>
<tr>
<td>fp0-fp31</td>
<td>Floating point registers 0 through 31</td>
</tr>
<tr>
<td>fpscr</td>
<td>Floating point status and control register</td>
</tr>
<tr>
<td>iar</td>
<td>Instruction address register</td>
</tr>
<tr>
<td>lr</td>
<td>Link register</td>
</tr>
<tr>
<td>mq</td>
<td>Multiply quotient</td>
</tr>
<tr>
<td>msr</td>
<td>Machine State register</td>
</tr>
<tr>
<td>r0-r31</td>
<td>General Purpose Registers 0 through 31</td>
</tr>
<tr>
<td>rtcl</td>
<td>Real Time clock (nanoseconds)</td>
</tr>
<tr>
<td>rtcu</td>
<td>Real Time clock (seconds)</td>
</tr>
<tr>
<td>s0-s15</td>
<td>Segment registers</td>
</tr>
<tr>
<td>sdr0</td>
<td>Storage description register 0</td>
</tr>
<tr>
<td>sdr1</td>
<td>Storage description register 1</td>
</tr>
<tr>
<td>srr0</td>
<td>Machine status save/restore 0</td>
</tr>
<tr>
<td>srr1</td>
<td>Machine status save/restore 1</td>
</tr>
<tr>
<td>tbl</td>
<td>Time base register, lower</td>
</tr>
<tr>
<td>tbu</td>
<td>Time base register, upper</td>
</tr>
<tr>
<td>tid</td>
<td>Transaction register (fixed point)</td>
</tr>
<tr>
<td>Register</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>xer</td>
<td>Exception register (fixed point)</td>
</tr>
</tbody>
</table>

Other special purpose registers that can be referenced, if they are supported on the hardware, include the following:

- sprg0
- sprg1
- sprg2
- sprg3
- pir
- fpecr
- ear
- pvr
- hid0
- hid1
- iabr
- dmiss
- imiss
- dcmp
- icmp
- hash1
- hash2
- rpa
- buscsr
- l2cr
- l2sr
- mmcr0
- mmcr1
- pmc1
- pmc2
- pmc3
- pmc4
- pmc5
- pmc6
- pmc7
- pmc8
- sia
- sda
Expressions

The KDB kernel debugger and kdb command can parse a limited set of expressions. Expressions can only contain symbols, hexadecimal constants, references to register or memory locations, and operators. Supported operators include the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>(</td>
<td>Parenthesis (order of operations)</td>
</tr>
<tr>
<td>@</td>
<td>Dereferencing</td>
</tr>
</tbody>
</table>

The dereference operator does the following:
- Indicates that the value at the location indicated by the next operand is to be used in the calculation of the expression.
  For example, @f000 indicates that the value at address 0x0000f000 should be used in evaluation of the expression.
- Allows access to the contents of a register.
  For example, @r1 references the contents of general purpose register 1. Recursive dereferencing is allowed. As an example, @@r1 references the value at the address pointed to by the value at the address contained in general purpose register 1.

The + and - operators have equal precedence. Likewise, the * / % and ^ operators have equal precedence with each other. Multiple operators with the same precedence are always evaluated from left to right in an expression. The following are examples:

<table>
<thead>
<tr>
<th>Valid Expressions</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>dw @r1</td>
<td>Displays data at the location pointed to by r1.</td>
</tr>
<tr>
<td>dw @r1+12</td>
<td>Displays data twelve bytes past the beginning of the open routine.</td>
</tr>
<tr>
<td>dw open</td>
<td>Displays data at the location pointed to by value at location pointed to by r1.</td>
</tr>
<tr>
<td>dw open+12</td>
<td>Displays data at the location pointed to by value at location pointed to by r1.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invalid Expressions</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>dw r1</td>
<td>Must include the at sign (@) to reference the contents of r1, if a symbol r1 existed, this would be valid.</td>
</tr>
</tbody>
</table>

User-defined variables

Both the KDB kernel debugger and the kdb command allow for user-defined variables. These variables can be used to provide a custom name for a memory address or an alias for a commonly used subcommand. After a user-defined variable is created, every occurrence of that variable in a subcommand is automatically replaced with the value assigned to the variable.
Variable substitution occurs before any other parsing of the subcommand. This allows a single variable to expand into multiple subcommand arguments. The `varset`, `varrm`, and `varlist` subcommands are used respectively for assigning, removing, and listing user-defined variables. The following is an example of how user-defined variables are used:

```kdb
KDB(0)> varset myvar kdb_avail
KDB(0)> dw myvar
<<dw kdb_avail>>
kdb_avail+000000: 00000001 00000001 00000000 00000004 000001C43 ...........L...C
KDB(0)> varset myvar kdb_avail 1
KDB(0)> dw myvar
<<dw kdb_avail 1>>
kdb_avail+000000: 00000000 ....
KDB(0)>
```

Any time a user variable expansion takes place at the debugger prompt, the expanded command line is printed between the `<<` and `>>` marks.

### Command line editing

Command line editing at the `KDB(0)>` or `(0)>` debugger prompt is supported and includes a history of recent commands. In addition, the command line supports several emacs and vi key bindings for editing text.

The `set` subcommand can be used to select the edit mode. The edit mode determines the set of key bindings that is currently active.

Regardless of which editing mode is used, the Ctrl+S and the Ctrl+Q key sequences are always available. The Ctrl+S key sequence pauses the debugger’s output to the screen and the Ctrl+Q key sequence causes the output to continue to resume the screen display.

### The emacs or gmacs editing mode

If the emacs or gmacs mode is active, the following key bindings are supported:

<table>
<thead>
<tr>
<th>Key Sequence</th>
<th>Associated Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl+F</td>
<td>Move the cursor one character forward.</td>
</tr>
<tr>
<td>Ctrl+B</td>
<td>Move the cursor one character backward.</td>
</tr>
<tr>
<td>Ctrl+A</td>
<td>Move the cursor to the beginning of the command line.</td>
</tr>
<tr>
<td>Ctrl+E</td>
<td>Move the cursor to the end of the command line.</td>
</tr>
<tr>
<td>Ctrl+P</td>
<td>Display the previous command in the history buffer.</td>
</tr>
<tr>
<td>Ctrl+N</td>
<td>Display the next command in the history buffer.</td>
</tr>
<tr>
<td>Ctrl+D</td>
<td>Delete the character at the cursor position.</td>
</tr>
<tr>
<td>Ctrl+U</td>
<td>Delete the entire command line.</td>
</tr>
<tr>
<td>Ctrl+T</td>
<td>In emacs mode, transpose the current and previous characters. In gmacs mode, transpose the previous two characters.</td>
</tr>
</tbody>
</table>

In addition the emacs and gmacs modes, allow a repeat count to be used with several of the above key sequences. If the Esc key is pressed followed by one or more numbers, and finally one of the above Ctrl key sequences is pressed, then the numbers following the Esc key are interpreted as a repeat count for the final Ctrl key sequence.
The vi editing mode

When the vi edit mode is active, the command prompt can be in either the vi text-input mode or the vi command mode. The command line starts in text-input mode where all typed characters become part of the text on the command line. Pressing the Esc key while in text-input mode switches your screen to the vi command mode. In the command mode, the debugger recognizes the following standard vi subcommands: l w W e E h b B | $ f F t T ; , k - j + G ? / n N . a i A s S R ~ I C D x X p P Y r y d c u and U.

Note: Any vi subcommands that begin with a colon are not supported.

For more information about vi subcommands, see vi subcommands in AIX 5L Version 5.3 Commands Reference, Volume 6.

Multiprocessor systems

On multiprocessor systems, entering the KDB kernel debugger stops all processors except the current processor running the debug program itself. On multiprocessor systems, the number in parentheses that is part of the prompt indicates the current processor. For example:

- For the following prompt, KDB(0)>-, the number 0 is contained in parentheses and is the current processor.
- For the following prompt, KDB(5)>-, the number 5 is contained in parentheses and is the current processor.

In addition to the change in the prompt for multiprocessor systems, there are also subcommands that are unique to these systems. For more information about the subcommands that can be used on multiprocessor systems, see Chapter 7, “Subcommand lists,” on page 29. The subcommands that are unique to multiprocessors are identified in the usage column.
Chapter 3. Viewing and modifying global data

Note: The demo and demokext programs are used in the examples in this section. The demokext_j variable, which is exported is used in the examples.

Global data can be accessed using several methods:

- **Method 1: Using the symbol name** demonstrates the simplest method of accessing global data. This is the primary method of accessing global data when using the KDB kernel debugger. The other methods are described to show alternatives and to allow the use of additional KDB subcommands in examples.

- **Method 2: Using the TOC and map file** on page 14 demonstrates accessing global data using the TOC and the map file. This method requires that the system is stopped in the KDB kernel debugger within a procedure of the kernel extension to be debugged. The address of the data for the demokext_j variable is calculated.

- **Method 3: Using the map file** on page 15 demonstrates a way to access global data using the map file, but without using the TOC. The address of the data for the demokext_j variable is calculated.

Before using any of the following examples, see “Loading the kernel extension” on page 443.

**Method 1: Using the symbol name**

Global variables within the KDB kernel debugger can be accessed directly by name. For example, the dw subcommand can be used to display the value of the demokext_j variable. If the demokext_j variable is an array, a specific value can be viewed by adding the appropriate offset (for example, dw demokext_j+20).

Access to individual elements of a structure is accomplished by adding the proper offset to the base address for the variable.

Note: The default prompt is KDB(0)>.

To view and modify global variables using the symbol name, do the following:

1. Display a word at the address of the demokext_j variable with the following command:
   dw demokext_j

   Because the kernel extension was just loaded, this variable should have a value of 99 and the KDB kernel debugger should display that value. The data displayed should be similar to the following:
   demokext_j+000000: 00000063 01304040 01304754 00000000 ...c.0@@.0GT....

2. Turn off symbolic name translation by typing the following:
   ns

3. To display the word at the address of the demokext_j variable, type the following:
   dw demokext_j

   With symbolic name translation turned off, the data displayed should be similar to the following:
   01304744: 00000063 01304040 01304754 00000000 ...c.0@@.0GT....

4. Turn symbolic name translation on by typing the following:
   ns

5. Modify the word at the address of the demokext_j variable by typing the following:
   mw demokext_j

   The KDB kernel debugger displays the current value of the word and waits for user input to change the value. The data displayed should be similar to the following:
   01304744: 00000063 =
Type a new value and press Enter. After a new value is entered, the next word of memory is displayed for possible modification. To end memory modification type a period (.) and press Enter. Type a value of 64 (100 decimal) for the first address, type a period and press Enter to end modification.

Method 2: Using the TOC and map file

Before you can locate the address of global data using the address of the TOC and the map file, the system must be stopped in the KDB kernel debugger within a routine of the kernel extension you want to debug. To do this, set a breakpoint within the kernel extension. For more information about setting a breakpoint, see Chapter 5, “Setting breakpoints,” on page 21.

When the KDB kernel debugger is invoked, general purpose register number 2 points to the address of the TOC. From the map file, the offset from the start of the table of contents (TOC) to the desired TOC entry can be calculated. Knowing this offset, and knowing the address at which the TOC starts, allows the address of the TOC entry for the desired global variable to be calculated. Then, the address of the TOC entry for the desired variable can be examined to determine the address of the data.

For example, assume that the KDB kernel debugger was invoked because of a breakpoint at line 67 of the demokext routine, and that the value for general purpose register number 2 is 0x01304754.

To find the address of the demokext_j variable, complete the following:
1. Calculate the offset from the beginning of the TOC to the TOC entry for the demokext_j variable. From the map file, the TOC starts at 0x0000010C and the TOC entry for the demokext_j variable is at 0x000000114. Therefore, the offset from the beginning of the TOC to the entry of interest is:
   $$0x00000114 - 0x0000010C = 0x00000008$$
2. Calculate the address of the TOC entry for the demokext_j variable. This is the current value of general purpose register 2 plus the offset calculated in the preceding step. The calculation is as follows:
   $$0x01304754 + 0x00000008 = 0x0130475C$$
3. Display the data at 0x0130475C. The data displayed is the address of the data for demokext_j.

To view and modify global data, do the following:
1. At the KDB(0) prompt, set a break at line 67 of the demokext routine by typing the following:
   ```
   b demokext+e0
   ```
   **Note:** Breaking at this location ensures that the KDB kernel debugger is invoked while within the demokext routines.
2. Obtain the value of General Purpose Register 2. You need that to determine the address of the TOC.
3. Exit the KDB kernel debugger by typing g on the command line.
4. Bring the demo program to the foreground and choose a selection. Choosing a selection causes the demokext routine to be called for configuration. Because a break was set, this causes the KDB kernel debugger to be invoked.
   **Note:** The prompt changes to a dollar sign ($).
5. Bring the demo program to the foreground by typing the following:
   ```
   fg
   ```
   **Note:** The prompt changes to ./demo.
6. Enter a value of 1 to select the option to increment the counters within the demokext kernel extension. This causes a break at line 67 of the demokext kernel extension and the prompt changes to KDB(0).
7. Display the general purpose registers by typing the following:
   ```
   dr
   ```
The data displayed should be similar to the following:

- \texttt{r0} : 0130411C
- \texttt{r1} : 2FF3B210
- \texttt{r2} : 01304754
- \texttt{r3} : 01304744
- \texttt{r4} : 0047B180
- \texttt{r5} : 0047B230
- \texttt{r6} : 000005FB
- \texttt{r7} : 00000000
- \texttt{r8} : 0x00000000
- \texttt{r9} : 000DD300
- \texttt{r10} : 00000000
- \texttt{r11} : 00000000
- \texttt{r12} : 013042F4
- \texttt{r13} : DEADBEEF
- \texttt{r14} : 0047B230
- \texttt{r15} : 2FF22D80
- \texttt{r16} : 2FF22D88
- \texttt{r17} : 00000000
- \texttt{r18} : 00000000
- \texttt{r19} : 0x000DD300
- \texttt{r20} : 00000000
- \texttt{r21} : 0x000DD300
- \texttt{r22} : 00000000
- \texttt{r23} : 00000000
- \texttt{r24} : 000DD300
- \texttt{r25} : 000DD300
- \texttt{r26} : 00000000
- \texttt{r27} : 000DD300
- \texttt{r28} : 000DD300
- \texttt{r29} : 000DD300
- \texttt{r30} : 000DD300
- \texttt{r31} : 0x01304648

Using the map, the offset to the TOC entry for the \textit{demokext.j} variable from the start of the TOC was 0x00000008. Adding this offset to the value displayed for \texttt{r2} indicates that the TOC entry of interest is at: 0x0130475C.

\textbf{Note:} The KDB kernel debugger can be used to perform the addition. In this case, the subcommand to use is \texttt{hcal \_\texttt{r2}+8}. For more information about the \texttt{hcal} subcommand, see \textit{hcal and dcal subcommands} on page 70.

8. Display the TOC entry for the \textit{demokext.j} variable by typing the following:
   \texttt{dw 0130475C}

   This entry contains the address of the data for the \textit{demokext.j} variable. The data displayed should be similar to the following:
   \texttt{TOC+000008: 01304744 000BCB34 00242E94 001E0518 .0GD...4.$......}

   The value for the first word displayed is the address of the data for the \textit{demokext.j} variable.

9. Display the data for the \textit{demokext.j} variable by typing the following:
   \texttt{dw 01304744}

   The displayed data should indicate that the value for the \textit{demokext.j} variable is still 0x0000064. This was set earlier because the breakpoint set was in the \textit{demokext} routine prior to incrementing the \textit{demokext.j} variable. The data displayed should be similar to the following:
   \texttt{demokext.j+000000: 00000064 01304040 01304754 00000000 ...d.0@@.0GT....}

10. Clear all breakpoints with the following command:
    \texttt{ca}

11. Exit the kernel debugger by typing \texttt{g} on the command line.

   \textbf{Note:} When you exit, the demo program is in the foreground and a prompt for the next option is displayed. The kernel extension is going to run and increment the \textit{demokext.j} variable. Next time it should have a value of 0x0000065.

12. Type the Ctrl+Z key sequence to stop the demo program. At this point, the prompt changes to a dollar sign ($).

13. Place the demo program in the background by typing the following:
    \texttt{bg}

\textbf{Method 3: Using the map file}

Unlike the procedure outlined in \textit{Method 2: Using the TOC and map file} on page 14, this method can be used at any time. This method requires the map file and the address at which the kernel extension was loaded.

\textbf{Note:} Because this method depends on how a kernel extension is loaded, this method might quit working if the procedure for loading a kernel extension is changed.

This method relies on the assumption that the address of a global variable can be found by using the following formula:
Addr of variable = Addr of the last function before the variable in the map + 
Length of the function + 
Offset of the variable

The following is a part of the map file for the demokext kernel extension:

```
 20 000005B8 000028 2 GL SD S17 <.fp_write> glink.s(/usr/lib/glink.o)
 21 000005B8  GL LD S18   .fp_write
 22 000005E0 000028 2 GL SD S19 <.fp_open> glink.s(/usr/lib/glink.o)
 23 000005E0  GL LD S20   .fp_open
 24 00000000 0000F9 3 RW SD S21 <$STATIC> demokext.c(demokext.o)
 25  E 000000FC 000004 2 RW SD S22 demokext_j demokext.c(demokext.o)
 26  * 00000100 00000C 2 DS SD S23 demokext demokext.c(demokext.o)
 27 0000010C 000000 2 TO SD S24 <TOC>
 28 0000010C 000004 2 TC SD S25 <$STATIC>
 29 00000110 000004 2 TC SD S26 <$system_configuration>
```

The last function in the .text section is at lines 22 and 23. The offset of this function from the map is 0x000005E0 (line 22, column 2). The length of the function is 0x000028 (Line 22, column 3). The offset of the demokext_j variable is 0x000000FC (line 25, column 2). So the offset from the load point value to the demokext_j variable is:

\[ 0x000005E0 + 0x000028 + 0x000000FC = 0x00000704 \]

Adding this offset to the load point value of the demokext kernel extension provides the address of the data for the demokext_j variable. Assuming a load point value of 0x01304040, this indicates that the data for the demokext_j variable is located at:

\[ 0x01304040 + 0x00000704 = 0x01304744 \]

To view global data, complete the following:

1. Activate KDB kernel debugger. Use the appropriate key sequence for your configuration. When this step is complete, you should see a KDB prompt.

2. Display the data for the demokext_j variable by typing the following:

   \[ \text{dw demokext+704} \]

   The 704 value is calculated from the map using the procedure listed above. This offset is then added to the load point of the demokext routine. The value for the demokext_j variable should now be 0x00000065. The data displayed should be similar to the following:

   \[ \text{demokext_j+000000: 00000065 01304040 01304754 00000000 ...e.0@@.0GT...} \]

   \[ \text{Note: There are numerous ways to find this address. For other methods, see \text{'Chapter 5, \"Setting breakpoints,\" on page 21.'}} \]

3. Exit the KDB kernel debugger by typing \text{g} on the command line and pressing Enter. The prompt changes to a dollar sign ($).

4. Bring the demo program to the foreground by typing \text{fg} and pressing Enter. The prompt changes to \text{./demo}.

5. Type \text{0} and press Enter to unload the demokext kernel extension and exit.
Chapter 4. Viewing stack traces

This topic describes:

- "Stack frame format"
- "Verbose stack output" on page 19

Note: The examples in this topic assume that the current process is the demonstration program that called the demokext kernel extension because there was a breakpoint set.

Stack frame format

To learn how to view and manipulate stack frame formats, perform the following steps:

1. Load the demokext kernel extension program. For directions, see "Loading the kernel extension" on page 443.

2. Display the stack for the current process, by typing stack and pressing Enter.

   The stack trace back displays the routines called and traces back through system calls. The displayed data should be similar to the following:

   thread+001800 STACK:
   [013042C0]write_log+00001C (10002040, 2FF3B258, 2FF3B2BC)
   [01304080]demokext+000070 (00000001, 2FF3B338)
   [001E3BF4]config_kmod+0000F0 (??, ??, ??)
   [001E3FA8]sysconfig+000140 (??, ??, ??)
   [000039D8].sys_call+000000 ()
   [10000570]main+000280 (??, ??)

3. To step forward four instructions, type s 4 and press Enter.

4. Reexamine the stack by typing stack and pressing Enter.

   It should now include the strlen call and should look similar to the following:

   thread+001800 STACK:
   [01304500]strlen+000000 ()
   [013042CC]write_log+000028 (10002040, 2FF3B258, 2FF3B2BC)
   [01304080]demokext+000070 (00000001, 2FF3B338)
   [001E3BF4]config_kmod+0000F0 (??, ??, ??)
   [001E3FA8]sysconfig+000140 (??, ??, ??)
   [000039D8].sys_call+000000 ()
   [10000570]main+000280 (??, ??)

5. If you do not see the strlen function call, continue stepping until it is displayed.

6. Toggle the KDB kernel debugger option to display the top 64 bytes for each stack frame by typing set display_stack_frames and pressing Enter.

7. Display the stack again with the display_stack_frames option turned on by typing stack and pressing Enter.

   The output should be similar to the following:

   thread+001800 STACK:
   [01304510]strlen+000000 ()
   [01304500]strlen+000000 (10002040, 2FF3B258, 2FF3B2BC)
   [01304080]demokext+000070 (00000001, 2FF3B338)
   [001E3BF4]config_kmod+0000F0 (??, ??, ??)
   [001E3FA8]sysconfig+000140 (??, ??, ??)
   [000039D8].sys_call+000000 ()
   [10000570]main+000280 (??, ??)

© Copyright IBM Corp. 1997, 2006
The displayed data can be interpreted using the diagram displayed in the [Subroutine Linkage](#) section of the *Assembler Language Reference* book.

8. Toggle the **display_stack_frames** option off by typing set display_stack_frames and pressing Enter.

9. Toggle the KDB kernel debugger option to display the registers saved in each stack frame by typing set display_stacked_regs and pressing Enter.

10. Display the stack again with the **display_stacked_regs** option activated by typing stack and pressing Enter.

The display should be similar to the following:

```c
thread+001800 STACK:
[01304510]strlen+000010 ()
[0130442C]write_log+000020 (10002040, 2FF3B258, 2FF3B28C)
    r30 : 00000000 r31 : 01304648
[013040B0]demokext+000070 (00000001, 2FF3B338)
    r30 : 00000000 r31 : 00000000
[001E3BF4]config_kmod+0000F0 (?, ?, ?)
    r30 : 040AO000 r31 : 00000000
[000039D8].sys_call+0000
[00000390].sys_call+000000 ()
[10000570]main+000280 (?, ?, ?)
[10000188]__start+000088 ()
```

11. Toggle the **display_stacked_regs** option off by typing set display_stacked_regs and pressing Enter.
Verbose stack output

To see more information about stack outputs, do the following:

1. Display the stack in raw format by typing `dw $r1 90` and pressing Enter:

   2. Clear all breakpoints by typing the following:

   3. Exit the kernel debugger by typing `g` on the command line. Upon exiting the debugger, the prompt from the demo program is displayed. The default prompt is `.demo`.

4. Enter a choice of `0` to unload the kernel extension and quit the KDB kernel debugger.
Chapter 5. Setting breakpoints

The KDB kernel debugger creates a table of breakpoints that it maintains. When a breakpoint is set, the debugger temporarily replaces the corresponding instruction with the trap instruction. The instruction overlaid by the breakpoint operates when you issue any subcommand that would cause that instruction to be initiated.

For more information on setting or clearing breakpoints, see Chapter 17, “Breakpoint and steps subcommands,” on page 115.

Setting a breakpoint is essential for debugging kernel extensions. The general steps for setting a breakpoint are the following:

1. Locate the assembler instruction corresponding to the C statement of the kernel system that you are debugging.
   - The process of locating the assembler instruction and obtaining its offset is explained in Chapter 3, “Viewing and modifying global data,” on page 13.
2. Get the offset of the assembler instruction from the listing.
3. Locate the address where the kernel extension is loaded.
4. Add the address of the assembler instruction to the address where kernel extension is loaded.
5. Set the breakpoint with the KDB \texttt{b} (break) subcommand.

Note: To continue with the demokext example, set a break at the C source line 67, which increments the demokext\_j variable. The list file indicates that this line starts at an offset of 0xE0.

The specific steps for setting a breakpoint are included in the following methods:

- “Method 1: Using the lke subcommand”
- “Method 2: Using the nm subcommand” on page 22
- “Method 3: Using the kmid pointer” on page 23
- “Method 4: Using the devsw subcommand” on page 23

Method 1: Using the lke subcommand

The KDB lke subcommand displays a list of loaded kernel extensions. To find the address of the modules for a particular extension use the KDB subcommand lke entry\_number, where entry\_number is the extension number of interest. A list of Process Trace Backs that shows the beginning addresses of routines contained in the extension is in the displayed data.

Note: The default prompt is KDB(0)>.

1. Determine the address where the kernel extension is loaded. For information about how to do this, see Chapter 3, “Viewing and modifying global data,” on page 13.
2. List all loaded extensions by typing lke on the command line.

The results should be similar to the following:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>FILE</th>
<th>FILESIZE</th>
<th>FLAGS</th>
<th>MODULE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>04E17F80</td>
<td>01303F00</td>
<td>000007F0</td>
<td></td>
<td>./demokext</td>
</tr>
<tr>
<td>04E17E80</td>
<td>0503A000</td>
<td>00000E88</td>
<td></td>
<td>/unix</td>
</tr>
<tr>
<td>04E17C00</td>
<td>04FA3000</td>
<td>00071B34</td>
<td></td>
<td>/usr/lib/drivers/nfs.ext</td>
</tr>
<tr>
<td>04E17A80</td>
<td>05021000</td>
<td>00000E88</td>
<td></td>
<td>/unix</td>
</tr>
<tr>
<td>04E17B00</td>
<td>01303898</td>
<td>00000348</td>
<td></td>
<td>/usr/lib/drivers/nfs_kdes.ext</td>
</tr>
<tr>
<td>04E17B00</td>
<td>04F96000</td>
<td>00000E34</td>
<td></td>
<td>/unix</td>
</tr>
<tr>
<td>04E17500</td>
<td>01301A10</td>
<td>0000217C</td>
<td></td>
<td>/etc/drivers/blockset64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1997, 2006
Enter the Ctrl-C key sequence to exit the KDB kernel debugger paging function. Pressing Enter displays the next page of data. Pressing the Spacebar displays the next line of data. The number of lines per page can be changed by typing `set screen_size nn` on the command line where `nn` is the number of lines per page.

3. List detailed information about the extension of interest.

The parameter to the `lke` subcommand is the slot number for the `./demokext` entry from the previous step. To display information for slot 1, type the following on the command line:

```
lke 1
```

The output from this command is similar to:

```
ADDRESS    FILE    FILESIZE    FLAGS    MODULE    NAME
1 04E17F80 01303F00 000007F0 00000272 ./demokext
le_flags....... TEXT    KERNELEX    DATATEXT    DATA    DATAEXISTS
le_next........ 04E17E80 le_fp......... 00000000
le_filename.... 04E17FD8 le_file....... 01303F00
le_filesize.... 000007F0 le_data....... 013045C8
le_tid......... 00000000 le_datasize.... 0000128
le_usecount.... 00000003 le_loadcount... 00000001
le_ndepend..... 00000001 le_maxdepend... 00000001
le_ule......... 0502E000 le_deferred.... 00000000
le_exports...... 0502E000 le_de......... 6C696263
le_searchlist.. 00000420 le_dlsusecount.. 00000000
le_dlindex..... 00002F6C le_lex......... 00000000
le_fh........... 00000000 le_depend.... 0 04E17FD4
TOC@........... 013046D4
```

From the PROCESS TRACE BACKS, you can see that the first instruction of `demokext` is at 01304040. The break for line 67 would be at this address plus E0.

4. Set the break at the desired location by typing the following:

```
b 01304040+e0
```

KDB displays the address at which the breakpoint is located.

5. Clear all breakpoints by typing the following:

```
ca
```

**Method 2: Using the nm subcommand**

If the kernel extension is not stripped, the KDB kernel debugger can be used to locate the address of the load point by name. For example, the `nm demokext` subcommand returns the address of the `demokext` routine after it is loaded. This address can then be used to set a breakpoint.

**Note:** The default prompt is `KDB(0)>`.

1. To translate a symbol to an effective address, type the following:

```
m demokext
```

The output is similar to the following:

```
Symbol Address : 01304040
TOC Address : 013046D4
```
The value of the **demokext** symbol is the address of the first instruction of the **demokext** routine. This value can be used to set a breakpoint.

2. Set the break at the desired location by typing the following:
   
   ```
b 01304040+e0
   ```

   KDB displays the address at which the breakpoint is set.

3. Display the word at the breakpoint by typing the following:
   
   ```
dw 01304040+e0
   ```

   The results are similar to the following:
   
   ```
   01304120: 80830000 30840001 90830000 809F0030 ....0........0
   ```

   This can be checked against the assembly code in the listing to verify that the break is set to the correct location.

4. Clear all breakpoints by typing the following:
   
   ```
   ca
   ```

---

**Method 3: Using the kmid pointer**

To locate the address of the entry point for a kernel extension, use the value of the **kmid** pointer returned by the `sysconfig(SYS_KLOAD)` subroutine when the kernel extension is loaded. The **kmid** pointer points to the address of the load point routine.

To get the address of the load point, print the **kmid** value during the `sysconfig` call from the configuration method. For example, use the **demo.c** module. Then start the KDB kernel debugger and display the value pointed to by the **kmid** pointer.

**Note:** The default prompt is KDB(0)>.

1. Display the memory at the address returned as the **kmid** pointer from the `sysconfig` subroutine, by typing the following:
   
   ```
dw 1304748
   ```

   KDB kernel debugger responds with something similar to:
   
   ```
   demokext+000000: 01304040 01304754 00000000 01304648 .000.0GT.....0FH
   ```

   The first word of data displayed is the address of the first instruction of the **demokext** routine. The data displayed is at the location `demokext+000000`. This corresponds to line 26 of the map presented earlier. However, `demokext+000000` and `.demokext+000000` are not the same address. The location `.demokext+000000` corresponds to line 10 of the map and is the address of the first instruction for the **demokext** routine.

2. Set the break at the location indicated from the previous command added to the offset to get to line 67 using the following command:
   
   ```
b 01304040+e0
   ```

   KDB kernel debugger responds with an indication of the address at which the breakpoint is set.

3. Clear all breakpoints by typing the following:
   
   ```
   ca
   ```

---

**Method 4: Using the devsw subcommand**

If the kernel extension is a device driver, use the KDB **devsw** subcommand to locate the desired address. The **devsw** subcommand lists all of the function addresses for the device driver that are in the dev switch table. Usually, the **config** subroutine is the load point routine. For example,
MAJ#010  OPEN  CLOSE  READ  WRITE
0123DE04  0123DE04  01230820  0123DA3C
10CTL  STRATEGY  TTY  SELECT
01230C090  0124DF0  00000000  00059774
CONFIG  PRINT  DUMP  MPX
0123EB0C8  0059774  00597774  00059774
REVOKE  DSPTR  SELPTR  OPTS
00059774  00000000  00000000  00000002

Note: The default prompt is KDB(0)>

To set a breakpoint, complete the following:

1. Display the device switch table for the first entry by typing the following:
   devsw 1

   The KDB kernel debugger devsw command displays data similar to the following:

   Slot address 50006040
   MAJ#001  OPEN  CLOSE  READ  WRITE
   .syopen  .nulldev  .syread  .sywrite
   IOCTL  STRATEGY  TTY  SELECT
   .syioctl  .nodev  00000000  .syselect
   CONFIG  PRINT  DUMP  MPX
   .nodev  .nodev  .nodev  .nodev
   REVOKE  DSPTR  SELPTR  OPTS
   .nodev  00000000  00000000  00000012

   Note: Because the demonstration program is not a device driver, this example uses the addresses of
   the first device driver in the device switch table and is not related in any way to the
   demonstration program.

2. Set a breakpoint at an offset of 0x20 from the beginning of the open routine for the first device driver in
   the device switch table by typing the following:
   b .syopen+20

   KDB kernel debugger displays the location of the break.

3. Clear all breakpoints by typing the following:
   ca

4. Turn off symbolic name translation by typing the following:
   ns

5. With symbolic name translation turned off, display the device switch table for the first device driver by
   typing the following:
   devsw 1

   The output is similar to the following:

   Slot address 50006040
   MAJ#001  OPEN  CLOSE  READ  WRITE
   00208858  00059750  002086D4  0020854C
   IOCTL  STRATEGY  TTY  SELECT
   00208290  0059774  00000000  00208224
   CONFIG  PRINT  DUMP  MPX
   00208858  00059750  002086D4  0020854C

6. Set a break at an offset of 0x20 from the beginning of the open routine for the first device driver in
   the device switch table by typing the following:
   b 00208858+20

   This sets the same break that was set at the beginning of this example. KDB displays the location of
   the break.

7. Toggle symbolic name translation on by typing the following:
ns

8. Clear all breaks by typing the following:
   ca

9. Exit the KDB kernel debugger and let the system resume normal operations by typing the following:
   g
Chapter 6. Using KDB kernel debugger to perform a trace

You can perform a system trace by starting kdb and the trace with the trcstart subcommand. Events are traced when you exits kdb command.

Upon subsequent re-entry, you can then view the trace output with the trace subcommand. The trace is stopped with the trcstop subcommand.

**Note:** This facility is only available through the KDB kernel debugger. It is not available through the kdb command.

If the system is in a working state, it is best to use the system trace facility and the trace subcommand. For more information about the trace command, see the [trace command](AIX 5L Version 5.3 Commands Reference, Volume 5)

The kdb command tracing capability is useful when the system is hung and does not respond to terminal input or when the system is initializing and the trace kernel extension is not loaded, because it can be used to determine where the kernel code is looping. It is is also helpful in early system-initialization debugging. For more information about the trace command, see the [trace command](AIX 5L Version 5.3 Commands Reference, Volume 5)

Only one trace event can be active at a time. A trace can be started from either the system trace facility at the shell prompt, or from the KDB kernel.

If a trace is started from the KDB kernel, and the system crashes, trace information can be extracted from the dump using the trcdead command. For more information about the trcdead command, see the [trcdead command](AIX 5L Version 5.3 Commands Reference, Volume 5).
Chapter 7. Subcommand lists

You can view an "Alphabetic list" of the subcommands or a "Task category list" on page 39.

The alphabetic list contains columns that identify the following:

- The name of the subcommand and any aliases for the subcommand. The name is linked to complete information about that subcommand.
- A brief description of the subcommand's function.
- A usage code that identifies when the subcommands can be used.
- Category in which the subcommands are grouped.

The task category list provides the following:

- Links from each task category to the section that lists the subcommands that are used for the task category.
- Links from each of the subcommands in the lists to the complete information for each subcommand. The information includes syntax, description, aliases and examples.

Alphabetic list

In the following table, the Usage column indicates when each subcommand can be used with the following codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>With <em>both</em> the KDB kernel debugger and the <em>kdb</em> command</td>
</tr>
<tr>
<td>C</td>
<td>Only with the <em>kdb</em> command</td>
</tr>
<tr>
<td>K</td>
<td>Only with the KDB kernel debugger</td>
</tr>
<tr>
<td>MP</td>
<td>An MP kernel (64-bit kernel or 32-bit MP kernel)</td>
</tr>
<tr>
<td>64</td>
<td>Only with 64-bit kernel</td>
</tr>
<tr>
<td>32</td>
<td>Only with 32-bit kernel</td>
</tr>
</tbody>
</table>

The following table shows the KDB Kernel Debug Program subcommands in alphabetic order:

<table>
<thead>
<tr>
<th>Subcommand, aliases</th>
<th>Functions</th>
<th>Usage</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Serves as a shell escape and provides a convenient way to run UNIX commands without leaving kdb</td>
<td>K</td>
<td>End user</td>
</tr>
<tr>
<td>ams</td>
<td>Display VMM address map entries</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>apt</td>
<td>Display VMM APT entries</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>b</td>
<td>Step on branch</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>b, brk</td>
<td>Sets or lists break points</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>bdev, wlm_bdev</td>
<td>Display <em>wlm</em> bio devices</td>
<td>B</td>
<td>WLM</td>
</tr>
<tr>
<td>bmblock, bmblk, bmb</td>
<td>Display Enhanced Journaled File System metadata block</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>bqueue, wlm_bq</td>
<td>Display <em>wlm</em> bio queues</td>
<td>B</td>
<td>WLM</td>
</tr>
<tr>
<td>bt</td>
<td>Set or list trace points</td>
<td>K</td>
<td>Debugger trace points</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>btac</td>
<td>Branch target</td>
<td>K</td>
<td>Branch target (IABR)</td>
</tr>
<tr>
<td>buffer, buf</td>
<td>Display buffer</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>buserr</td>
<td>PCI bus error injection</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>businfo</td>
<td>Display structure businfo</td>
<td>B</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>c, cl</td>
<td>Clear break point</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>ca</td>
<td>Clear all break points</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>cat</td>
<td>Clear all trace points</td>
<td>K</td>
<td>Debugger trace points</td>
</tr>
<tr>
<td>cbtac</td>
<td>Clear branch target</td>
<td>K</td>
<td>Branch target (IABR)</td>
</tr>
<tr>
<td>cdt</td>
<td>Display cdt</td>
<td>C</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>check</td>
<td>Run consistency checkers on kernel data structures</td>
<td>B</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>cla, class</td>
<td>Display wlm class</td>
<td>B</td>
<td>WLM</td>
</tr>
<tr>
<td>clk, cpl</td>
<td>Display complex lock</td>
<td>B</td>
<td>Locks</td>
</tr>
<tr>
<td>conv</td>
<td>Base conversion</td>
<td>B</td>
<td>Leaving</td>
</tr>
<tr>
<td>cpu</td>
<td>Switch to cpu</td>
<td>B, MP</td>
<td>Changing context</td>
</tr>
<tr>
<td>cr, crid</td>
<td>Display crid table</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>cred</td>
<td>Display credentials structure</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>ct</td>
<td>Clear trace point</td>
<td>K</td>
<td>Debugger trace points</td>
</tr>
<tr>
<td>ctctrl</td>
<td>Display information on Component Trace (CT)</td>
<td>B</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>ctx, context</td>
<td>Switch to KDB context</td>
<td>B, MP</td>
<td>Changing context</td>
</tr>
<tr>
<td>cupboard</td>
<td>Display NFS cupboard</td>
<td>B</td>
<td>Display NFS information</td>
</tr>
<tr>
<td>cw</td>
<td>Clear watch</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>d, dump</td>
<td>Display byte data</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>dbat</td>
<td>Display dbats</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>dbgopt</td>
<td>Enable or disable debug options</td>
<td>K</td>
<td>End user</td>
</tr>
<tr>
<td>dc, dis</td>
<td>Display code</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>dcal</td>
<td>Calculate or convert a decimal expression</td>
<td>B</td>
<td>Calculator / converter</td>
</tr>
<tr>
<td>dd</td>
<td>Display double word data</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddpb</td>
<td>Display device byte</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddpd</td>
<td>Display device double word</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddph</td>
<td>Display device half word</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddpw</td>
<td>Display device word</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>ddvb, diob</td>
<td>Display device byte</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddvd, diod</td>
<td>Display device double word</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddvh, dioh</td>
<td>Display device half word</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>ddvw, diow</td>
<td>Display device word</td>
<td>K</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>debug</td>
<td>Enable or disable debug</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>devsw, dev</td>
<td>Display devsw table</td>
<td>B</td>
<td>Display miscellaneous kernel data structures</td>
</tr>
<tr>
<td>devnode, devno</td>
<td>Display devnode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>di, decode</td>
<td>Decode the given instruction</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>dla</td>
<td>Checks the system for deadlocks and displays details on threads waiting on locks</td>
<td>B, 64</td>
<td>Locks</td>
</tr>
<tr>
<td>dlk</td>
<td>Display dist lock</td>
<td>B, 64</td>
<td>Locks</td>
</tr>
<tr>
<td>dnlc, ncache</td>
<td>Display name cache</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>dp</td>
<td>Display byte data</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>dpc</td>
<td>Display code</td>
<td>B</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>dpcib</td>
<td>Display PCI configuration space in bytes</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>dpcih</td>
<td>Display PCI configuration space in half words</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>dpciw</td>
<td>Display PCI configuration space in words</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>dpd</td>
<td>Display double word data</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>dpw</td>
<td>Display word data</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>dr</td>
<td>Display registers</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>drlist</td>
<td>Display DRlist</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>drvars, drv</td>
<td>Display DRvars</td>
<td>B, MP</td>
<td>Display miscellaneous kernel data structures</td>
</tr>
<tr>
<td>dtree, dt</td>
<td>Display Enhanced Journaled File System dtree</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>dw</td>
<td>Display word data</td>
<td>B</td>
<td>Memory register display and decode</td>
</tr>
<tr>
<td>e, q, g</td>
<td>Exit</td>
<td>B</td>
<td>Leaving</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>errpt</td>
<td>Display error log entries</td>
<td>B</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>exp</td>
<td>List export tables</td>
<td>B</td>
<td>Loader</td>
</tr>
<tr>
<td>ext, stack, where</td>
<td>Stack frame trace</td>
<td>B</td>
<td>Common basic display</td>
</tr>
<tr>
<td>fbuffer, fb</td>
<td>Display freelist</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>fifono, ffonode</td>
<td>Display fifonode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>file</td>
<td>Display file</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>find, findp</td>
<td>Find symbolic pattern</td>
<td>B</td>
<td>Memory search and extract</td>
</tr>
<tr>
<td>frameset, fres</td>
<td>Display frame sets</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>free</td>
<td>Count and display free frames</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>freelist</td>
<td>Display free list</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>gfs</td>
<td>Display gfs</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>gnode, gno</td>
<td>Display gnode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>gt</td>
<td>Go until address</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>h, ?, help</td>
<td>Help</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>halt</td>
<td>Halt the machine</td>
<td>K</td>
<td>Leaving</td>
</tr>
<tr>
<td>hbuffer, hb</td>
<td>Display buffer hash</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>hcal, cal</td>
<td>Calculate or convert a hexadecimal expression</td>
<td>B</td>
<td>Calculator / converter</td>
</tr>
<tr>
<td>hdnlc, hncache</td>
<td>Display hash and ncache</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>heap, hp</td>
<td>Display kernel heap</td>
<td>B</td>
<td>Display memory allocator information</td>
</tr>
<tr>
<td>hinode, hino</td>
<td>Display inodehash</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>his, hi, hist</td>
<td>Print history</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>hnode, hno</td>
<td>Display hnodehash</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>hvnc, hvcache</td>
<td>Display hash, vcache</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>ibat</td>
<td>Display ibats</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>icache, fino</td>
<td>Display icache list</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>ifnet</td>
<td>Display interface</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>inode, ino</td>
<td>Display inode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>inode2, i2</td>
<td>Display Enhanced Journaled File System inode</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>intr</td>
<td>Display int handler</td>
<td>B</td>
<td>Display miscellaneous kernel data structures</td>
</tr>
<tr>
<td>ipc</td>
<td>Display IPC information</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>1pl</td>
<td>Display IPL process information</td>
<td>B</td>
<td>Display miscellaneous kernel data structures</td>
</tr>
<tr>
<td>j2, jfs2</td>
<td>Display Enhanced Journaled File System buffer data</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>j2logbuf</td>
<td>Display JFS2 log buffer structure</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>j2logx</td>
<td>Display Enhanced Journaled File System logx structure</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>j2no, jfs2node</td>
<td>Display jfs2node</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>kfset, kfs</td>
<td>Display the kdm fset cache data structure</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>kmbucket, bucket</td>
<td>Display kmembuckets</td>
<td>B</td>
<td>Display memory allocator information</td>
</tr>
<tr>
<td>kmstats</td>
<td>Display kmemstats</td>
<td>B</td>
<td>Display memory allocator information</td>
</tr>
<tr>
<td>ksp</td>
<td>Display KSP region information</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>kvn, kvnode</td>
<td>Display kdm vnode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>lastbackt</td>
<td>Display lastbackt</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>lb, lbrk</td>
<td>Sets or lists local breakpoints</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>lbtac</td>
<td>Display local branch target</td>
<td>K</td>
<td>Branch target (IABR)</td>
</tr>
<tr>
<td>lc, lcl</td>
<td>Clear local breakpoints</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>lcbtac</td>
<td>Clear local branch target</td>
<td>K</td>
<td>Branch target (IABR)</td>
</tr>
<tr>
<td>lcw</td>
<td>Clear local watch</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>lk</td>
<td>Display lock_t lock</td>
<td>B</td>
<td>Locks</td>
</tr>
<tr>
<td>lke</td>
<td>List loaded extensions</td>
<td>B</td>
<td>Loader</td>
</tr>
<tr>
<td>lle</td>
<td>List loader entries</td>
<td>B</td>
<td>Loader</td>
</tr>
<tr>
<td>lka, lockanch tblk</td>
<td>Display VMM lock anchor or tblock</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>lkh, lockhash</td>
<td>Display VMM lock hash</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>lk, lockword</td>
<td>Display VMM lock word</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>lq, lockq</td>
<td>Display lock queues</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>lrustate, lru</td>
<td>Display the lru daemon control variables</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>lvol</td>
<td>Display logical volume</td>
<td>B</td>
<td>Display storage subsystem information</td>
</tr>
<tr>
<td>lwrt</td>
<td>Local stop on read data</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>lww</td>
<td>Local stop on read/write data</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>lwwp</td>
<td>Local stop on write data</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>m</td>
<td>Modify sequential bytes</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mbuf</td>
<td>Display mbuf</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>md</td>
<td>Modify sequential double word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdbat</td>
<td>Modify dbats</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>mdcb</td>
<td>Modify device byte</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdcp</td>
<td>Modify device double word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdph</td>
<td>Modify device half</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdpw</td>
<td>Modify device word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdvb, miob</td>
<td>Modify device byte</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdvd, miold</td>
<td>Modify device double word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdvh, miowh</td>
<td>Modify device half</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mdvw, miow</td>
<td>Modify device word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>meml, memlock</td>
<td>Displays information about the memory lock entries</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>mempool, memp</td>
<td>Display memory pools</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>mibat</td>
<td>Modify ibats</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>mp</td>
<td>Modify sequential bytes</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mpcib</td>
<td>Modify PCI configuration space in bytes</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>mpch</td>
<td>Modify PCI configuration space in half words</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>mpciw</td>
<td>Modify PCI configuration space in words</td>
<td>K</td>
<td>PCI cfg space and I/O debugging</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>mpd</td>
<td>Modify sequential double word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mpw</td>
<td>Modify sequential word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mr</td>
<td>Modify registers</td>
<td>B</td>
<td>Memory modification</td>
</tr>
<tr>
<td>mslib</td>
<td>Modify SLB entry</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>mst</td>
<td>Display MST area</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>mtrace</td>
<td>Display information on the Lightweight Memory Trace (LMT)</td>
<td>B</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>mw</td>
<td>Modify sequential word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>n, nexti</td>
<td>Next instruction</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>ndd</td>
<td>Display network and device driver statistics</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>netm</td>
<td>Display the <code>net_malloc</code> event records</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>netstat</td>
<td>Display network status</td>
<td>C</td>
<td>Network</td>
</tr>
<tr>
<td>nsdbg</td>
<td>Display ns_alloc and free event records stored in the kernel.</td>
<td>C</td>
<td>Network</td>
</tr>
<tr>
<td>nm</td>
<td>Translate symbol to an effective address</td>
<td>B</td>
<td>Namelists and symbols</td>
</tr>
<tr>
<td>ns</td>
<td>No symbol mode (toggle)</td>
<td>B</td>
<td>Namelists and symbols</td>
</tr>
<tr>
<td>pbuf</td>
<td>Display physical buf</td>
<td>B</td>
<td>Display storage subsystem information</td>
</tr>
<tr>
<td>pdt</td>
<td>Display VMM paging device table</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>pfdt</td>
<td>Display VMM control variables</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>pft</td>
<td>Display VMM PFT entries</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>pgbuf</td>
<td>Display Enhanced Journaled File System pager buffer</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>pgobj</td>
<td>Display Enhanced Journaled File System pagerObject</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>pile</td>
<td>Display pile</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>pnda</td>
<td>Display PNDA area</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>ppda</td>
<td>Display per processor data area</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>ppd, pr_pid</td>
<td>Display displays all process entries for a process</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>pr, print</td>
<td>Print a formatted structure at an address</td>
<td>B</td>
<td>Common basic display</td>
</tr>
<tr>
<td>pta</td>
<td>VMM PTA segment</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>ptid, pr_tid</td>
<td>Display displays the process table entry for a specific process</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>pte</td>
<td>VMM PTE entries</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>pvlist, pvt</td>
<td>Display VMM PVT and PVLIST entries</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>pvol</td>
<td>Display physical volume</td>
<td>B</td>
<td>Display storage subsystem information</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>r, return</td>
<td>Go to end of function</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>reboot</td>
<td>Reboot the machine</td>
<td>K</td>
<td>Leaving</td>
</tr>
<tr>
<td>rmap</td>
<td>Display VMM RMAP</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>rmst</td>
<td>Remove symbol table</td>
<td>B</td>
<td>Loader</td>
</tr>
<tr>
<td>route</td>
<td>Display route</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>rq, runq</td>
<td>Display run queues</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>rqi, rqa</td>
<td>Display RQ information</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>rtentry</td>
<td>Display rentry structure</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>rtipc</td>
<td>Display RT IPC information</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>rtipc</td>
<td>Display RT IPCD information</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>rules, rule</td>
<td>Display wlm rules</td>
<td>B</td>
<td>WLM</td>
</tr>
<tr>
<td>runcpu</td>
<td>Allows any other kdb subcommand to be automatically run</td>
<td>B</td>
<td>Changing context</td>
</tr>
<tr>
<td>rvsid</td>
<td>Display reserved vsid information</td>
<td>B, 64</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>rxnode</td>
<td>Display radix_node structure</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>s, stepi</td>
<td>Single step</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>S</td>
<td>Step on block or blockr</td>
<td>K</td>
<td>Breakpoints and steps</td>
</tr>
<tr>
<td>scb</td>
<td>Display VMM segment control blocks</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>scdb, scdisk</td>
<td>Display scdisk</td>
<td>B</td>
<td>Display storage subsystem information</td>
</tr>
<tr>
<td>segst64</td>
<td>Display VMM SEGSTATE</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>set, setup</td>
<td>Display or update kdb toggles</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>slab</td>
<td>Display slab</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>slb</td>
<td>Display SLB entry</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>slik, spl</td>
<td>Display simple lock</td>
<td>B</td>
<td>Locks</td>
</tr>
<tr>
<td>sock</td>
<td>Display socket</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>sockcup</td>
<td>Display NFS sockcup</td>
<td>B</td>
<td>Display NFS information</td>
</tr>
<tr>
<td>sockinfo, si</td>
<td>Display socket info by address</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>sockpint</td>
<td>Display NFS sockcup</td>
<td>B</td>
<td>Display NFS information</td>
</tr>
<tr>
<td>specnode, specno</td>
<td>Display specnode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>sr64</td>
<td>Display VMM segment region</td>
<td>B, MP</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>start</td>
<td>Start cpu</td>
<td>K, MP</td>
<td>CPU start and stop</td>
</tr>
<tr>
<td>stat</td>
<td>System status messages</td>
<td>B</td>
<td>Common basic display</td>
</tr>
<tr>
<td>status</td>
<td>Processor status</td>
<td>B</td>
<td>Common basic display</td>
</tr>
<tr>
<td>si</td>
<td>Store one address word in memory</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>stbl</td>
<td>List loaded symbol tables</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>stc</td>
<td>Store one address byte in memory</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>ste</td>
<td>Display VMM STAB</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>sth</td>
<td>Store one half-word in memory address half-word</td>
<td>K</td>
<td>Memory modification</td>
</tr>
<tr>
<td>stop</td>
<td>Stop cpu</td>
<td>KMP</td>
<td>CPU start and stop</td>
</tr>
<tr>
<td>svcxprt</td>
<td>Display NFS SVCXPRT</td>
<td>B</td>
<td>Display NFS information</td>
</tr>
<tr>
<td>svmon</td>
<td>Display information about the memory and paging space usage on a per process basis</td>
<td>C</td>
<td>Display context information</td>
</tr>
<tr>
<td>svwhat</td>
<td>Display VMM SWHAT entries</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>sw, switch</td>
<td>Switch to thread</td>
<td>B</td>
<td>Changing context</td>
</tr>
<tr>
<td>symptom</td>
<td>Display symptom string for a dump</td>
<td>C</td>
<td>Common basic display</td>
</tr>
<tr>
<td>tcb</td>
<td>Display TCBs</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>tcpcb</td>
<td>Display TCP CB</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>tcpdbg</td>
<td>Displays the tcp_debug structures</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>test, ()</td>
<td>Displays bt condition</td>
<td>K</td>
<td>Debugger trace points</td>
</tr>
<tr>
<td>time</td>
<td>Display elapsed time</td>
<td>K</td>
<td>Time</td>
</tr>
<tr>
<td>tpid, th_pid</td>
<td>Display displays all thread entries for a process</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>tr</td>
<td>Translate to real address</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>trace</td>
<td>Display trace buffer</td>
<td>B</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>trb</td>
<td>Display system timer request blocks</td>
<td>B</td>
<td>Time</td>
</tr>
<tr>
<td>trcstart</td>
<td>Starts the system trace</td>
<td>K</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>trcstop</td>
<td>Stops the system trace</td>
<td>K</td>
<td>System trace, dump, and error log</td>
</tr>
<tr>
<td>tree</td>
<td>Display Enhanced Journaled File System tree</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>ts</td>
<td>Translate eaddr to symbol</td>
<td>B</td>
<td>Namelists and symbols</td>
</tr>
<tr>
<td>ttid, th_tid</td>
<td>Display displays the thread table entry for a specific thread</td>
<td>B</td>
<td>Display context information</td>
</tr>
<tr>
<td>tv</td>
<td>Display MMU translation</td>
<td>B</td>
<td>Address translation</td>
</tr>
<tr>
<td>txblock, txblk</td>
<td>Display Enhanced Journaled File System txBlock</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>txblocki, txblki</td>
<td>Display Enhanced Journaled File System index of txBlock</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>txlock, txlck</td>
<td>Display Enhanced Journaled File System txLock</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>udb</td>
<td>Display UDBs</td>
<td>B</td>
<td>Network</td>
</tr>
<tr>
<td>var</td>
<td>Display var</td>
<td>B</td>
<td>Display miscellaneous kernel data structures</td>
</tr>
<tr>
<td>varlist</td>
<td>List user variables</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>Subcommand, aliases</td>
<td>Functions</td>
<td>Usage</td>
<td>Category</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>varrm, unalias</td>
<td>Remove user variable</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>varset, alias</td>
<td>Define a user variable</td>
<td>B</td>
<td>End user</td>
</tr>
<tr>
<td>vfs, mount</td>
<td>Display vfs</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>vmdmap</td>
<td>VMM disk map</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmlocks</td>
<td>VMM spin locks</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmaddr</td>
<td>VMM Addresses</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmbufst</td>
<td>Display dump buffer structures</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmint</td>
<td>Display VMM vmintervals information</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmker</td>
<td>Display VMM kernel segment data</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmlocks, vmlock, vl</td>
<td>Display VMM spin locks</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmlog</td>
<td>Display VMM error log</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmpool</td>
<td>Display VMM resource pools</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmstat</td>
<td>Display VMM statistics</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmhrpio</td>
<td>Display THRPGIO commands</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vmwait</td>
<td>Display VMM wait status</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vnc, vcache</td>
<td>Display vnode cache</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>vnode, vno</td>
<td>Display vnode</td>
<td>B</td>
<td>Display general file system and Journal File System information</td>
</tr>
<tr>
<td>volgrp</td>
<td>Display volume group</td>
<td>B</td>
<td>Display storage subsystem information</td>
</tr>
<tr>
<td>vrld</td>
<td>Display VMM reload xlate table</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vsidd, sidd</td>
<td>Display VSID dump</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>vsidm, sidm</td>
<td>Display VSID alter</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
<tr>
<td>which</td>
<td>Display name of kernel source file</td>
<td>C</td>
<td>Namelists and symbols</td>
</tr>
<tr>
<td>wr</td>
<td>Stop on read data</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>wrw</td>
<td>Stop on r/w data</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>ww</td>
<td>Stop on write data</td>
<td>K</td>
<td>Watch DABR</td>
</tr>
<tr>
<td>xtree, xt</td>
<td>Display Enhanced Journaled File System xtree</td>
<td>B</td>
<td>Display Enhanced Journaled File System-specific file system information</td>
</tr>
<tr>
<td>xmalloc, xm</td>
<td>Display heap debug</td>
<td>B</td>
<td>Display memory allocator information</td>
</tr>
<tr>
<td>zproc</td>
<td>Display VMM zeroing kproc</td>
<td>B</td>
<td>Display VMM information</td>
</tr>
</tbody>
</table>
Task category list

The categories in which the subcommands are grouped are as follows:

- Chapter 25, “Address translation subcommands,” on page 241
- Chapter 20, “Branch target subcommands,” on page 137
- Chapter 17, “Breakpoint and steps subcommands,” on page 115
- Chapter 10, “Changing context subcommands,” on page 59
- Chapter 11, “Calculator and converter subcommands,” on page 69
- Chapter 13, “Basic display subcommands,” on page 75
- Chapter 12, “CPU start and stop subcommands,” on page 73
- Chapter 27, “Display context information subcommands,” on page 261
- Chapter 30, “Display general and Journal File System (JFS) information subcommands,” on page 311
- Chapter 31, “Display Enhanced Journaled File System information subcommands,” on page 349
- Chapter 29, “Display memory allocation information subcommands,” on page 297
- Chapter 23, “Display kernel data structures subcommands,” on page 155
- Chapter 32, “Display NFS information subcommands,” on page 371
- Chapter 28, “Display storage subsystem information subcommands,” on page 287
- Chapter 24, “Display VMM subcommands,” on page 165
- Chapter 8, “End user subcommands,” on page 41
- Chapter 18, “Debugger trace points subcommands,” on page 127
- Chapter 9, “Leaving kdb subcommands,” on page 55
- Chapter 26, “Loader subcommands,” on page 251
- Chapter 35, “Lock subcommands,” on page 397
- Chapter 16, “Memory modification subcommands,” on page 107
- Chapter 14, “Memory register display and decode subcommands,” on page 89
- Chapter 15, “Memory search and extract subcommands,” on page 101
- Chapter 21, “Namelist and symbols subcommands,” on page 141
- Chapter 36, “Network subcommands,” on page 401
- Chapter 22, “PCI configuration space and I/O debugging subcommands,” on page 145
- Chapter 34, “System trace, dump and error log subcommands,” on page 383
- Chapter 33, “Time subcommands,” on page 377
- Chapter 19, “Watch DABR subcommands,” on page 133
- Chapter 37, “Workload Manager (WLM) subcommands,” on page 433
Chapter 8. End user subcommands

The subcommands in this category explain how category help works, list and set kdb command toggles, and create, display and remove user-defined variables. These subcommands include the following:

- **h**
- **set**
- **dbgopt**
- **varset**
- **varlist**
- **varrm**
- **his**
- **debug**
- **!**
h subcommand

Purpose
The h subcommand displays a list of all available subcommands in the debugger. When run with a parameter, this list is restricted to only a particular category of subcommands. The list of categories is:

- Chapter 25, “Address translation subcommands,” on page 241
- Chapter 20, “Branch target subcommands,” on page 137
- Chapter 17, “Breakpoint and steps subcommands,” on page 115
- Chapter 10, “Changing context subcommands,” on page 59
- Chapter 11, “Calculator and converter subcommands,” on page 69
- Chapter 13, “Basic display subcommands,” on page 75
- Chapter 12, “CPU start and stop subcommands,” on page 73
- Chapter 27, “Display context information subcommands,” on page 261
- Chapter 30, “Display general and Journal File System (JFS) information subcommands,” on page 311
- Chapter 31, “Display Enhanced Journaled File System information subcommands,” on page 349
- Chapter 29, “Display memory allocation information subcommands,” on page 297
- Chapter 23, “Display kernel data structures subcommands,” on page 155
- Chapter 32, “Display NFS information subcommands,” on page 371
- Chapter 28, “Display storage subsystem information subcommands,” on page 287
- Chapter 24, “Display VMM subcommands,” on page 165
- Chapter 8, “End user subcommands,” on page 41
- Chapter 18, “Debugger trace points subcommands,” on page 127
- Chapter 9, “Leaving kdb subcommands,” on page 55
- Chapter 26, “Loader subcommands,” on page 251
- Chapter 35, “Lock subcommands,” on page 397
- Chapter 16, “Memory modification subcommands,” on page 107
- Chapter 14, “Memory register display and decode subcommands,” on page 89
- Chapter 15, “Memory search and extract subcommands,” on page 101
- Chapter 21, “Namelist and symbols subcommands,” on page 141
- Chapter 36, “Network subcommands,” on page 401
- Chapter 22, “PCI configuration space and I/O debugging subcommands,” on page 145
- Chapter 34, “System trace, dump and error log subcommands,” on page 383
- Chapter 33, “Time subcommands,” on page 377
- Chapter 19, “Watch DABR subcommands,” on page 133
- Chapter 37, “Workload Manager (WLM) subcommands,” on page 433

Syntax
h [topic]

Parameters

*topic* Specifies the name, or partial name, of a particular help category. If more than one category name matches the topic, only the first matching category and its subcommands are displayed.
## Aliases

?, help

### Example

The following is an example of how to use the help alias for the h subcommand:

```
KDB(0)> help user
```

<table>
<thead>
<tr>
<th>CMD</th>
<th>ALIAS</th>
<th>ALIAS</th>
<th>FUNCTION</th>
<th>ARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>?</td>
<td>help</td>
<td>help</td>
<td>[topic]</td>
</tr>
<tr>
<td></td>
<td>set</td>
<td>setup</td>
<td>display/update kdb toggles</td>
<td>[toggle]</td>
</tr>
<tr>
<td></td>
<td>dbgopt</td>
<td>alias</td>
<td>enable/disable debug options</td>
<td></td>
</tr>
<tr>
<td></td>
<td>varset</td>
<td>alias</td>
<td>define a user variable</td>
<td>var value</td>
</tr>
<tr>
<td></td>
<td>varlist</td>
<td></td>
<td>list user variables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>varrm</td>
<td>unalias</td>
<td>remove user variable</td>
<td>var</td>
</tr>
<tr>
<td></td>
<td>his</td>
<td>hi</td>
<td>print history</td>
<td>[?][count]</td>
</tr>
<tr>
<td></td>
<td>debug</td>
<td></td>
<td>enable/disable debug</td>
<td>[?]</td>
</tr>
</tbody>
</table>

KDB(0)>
set subcommand

Purpose
The set subcommand lists and sets kdb toggles.

Syntax
set [toggle [value]]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toggle</td>
<td>Identifies the option to be toggled or set by decimal number or name.</td>
</tr>
<tr>
<td>value</td>
<td>Indicates the decimal number or expression to be set for an option.</td>
</tr>
</tbody>
</table>

Note: Some toggles allow the value to be omitted. In that case, the set subcommand cycles the toggle through all of its possible settings.

The values that are valid for the KDB Kernel Debugger and the kdb command are the following:

- **no_symbol** Supresses symbol name lookup when addresses are displayed.
- **mst_wanted** Displays all mst items in the stack trace subcommand each time an interrupt is detected in the stack. For a shorter display, disable this toggle.
- **screen_size** Changes the integrated more prompt window size.
- **power_pc_syntax** Displays PowerPC platform-based instruction mnemonics when enabled (See the 92 and 94 subcommands). Displays the old POWER™ family mnemonics when disabled.
- **origin** Sets the origin variable to the value of the specified expression. Origins are used to match addresses with assembly language listings. Assembly language listings express addresses as offsets from the start of the file.
- **unix_symbols_start_from** Indicates the lowest effective address from which symbol search is started. To force other values to be displayed in hexadecimal, set this toggle.
- **hexadecinal_wanted** Applies to thread and process subcommand. It is possible to have information in decimal form.
- **screen_previous** Applies to the memory display subcommands, such as d and dw. To repeat the last memory display subcommand, press Enter at an empty kdb prompt. If screen_previous is set to false, memory is displayed at the next higher address. If screen_previous is set to true, memory is displayed at the next lower address.
- **display_stack_frames** Applies to f subcommand. When it is true, the f subcommand prints a part of the stack in binary mode.
- **display_stacked_regs** Applies to f subcommand. When it is true, the f subcommand prints register values saved in the stack.
- **64_bit** Prints 64-bit registers on 64-bit architecture. By default, only 32-bit formats are printed.
- **ldr_segs_wanted** Toggles interpretation of effective addresses in segment 11 (0xbxxxxxxx) and segment 13 (0xdxxxxxxx) off and on as references to loader data.
- **trace_back_lookup** Processes trace back information on user code (text or shared-lib) and kernext code. It can be used to see function names. By default, it is not set.
- **scroll** Enables or disables the integrated more prompt.
- **edit** Provides command line editing features similar to those provided by the Korn shell. The mode specified provides editing features similar to editors, such as vi, emacs, and gmacro.

For example, to turn on vi-style command line editing, type the following at the kdb prompt:

```
set edit vi
```

- **default_xmalloc_heap** Specifies the default heap for the xmalloc subcommand. If this option is 0, the xmalloc subcommand uses the kernel heap.
The values that apply only to the kdb command are the following:

logfile
Enables or disables logging for a specified log file name. If logfile is invoked without a parameter specifying a file name, logging is disabled.

loglevel
Allows you to choose the granularity level of logging. Valid choices are the following:

- 0 off
- 1 Log commands only
- 2 Log commands and output. This is the default.

auto_screen_size
Enables tracking of terminal size changes and automatic updates of screen_size as necessary. If you manually assign a value to screen_size, the auto_screen_size toggle will become false.

The options that apply only to the KDB kernel debugger are the following:

emacs_window
Toggles suppression of extra line feeds for running under emacs.

local_breakpoint_attach
Toggles to choose whether local breakpoints are thread or CPU based. By default, on POWER RS1, local breakpoints are CPU-based, and on the POWER-based platform they are thread-based.

Note: This toggle must be accessed using the option number. It cannot be toggled by name.

kdb_stop_all_cpu
Toggles to select whether all processors or a single processor stops when the KDB kernel debugger is invoked.

tweq_r1_r1
Causes the KDB kernel debugger to break on the tweq r1, r1 instruction. This is the trap instruction reserved for entering LLDB.

kext_IF_active
Toggles to disable and enable subcommands added to the KDB kernel debugger through kernel extensions. By default, all subcommands registered by kernel extensions are active.

IPI_enable
Toggles to control how the KDB kernel debugger notifies other processors to stop when the KDB stops all processors value is true. If the IPI_enable value is true, the KDB kernel debugger uses inter-processor interrupts. If IPI_enable is false, the decremener interrupt is used.

no_brkpt_warning
Controls whether the KDB kernel debugger prints warning messages when it ignores certain breakpoints, for example, a context mismatch. If the no_brkpt_warning value is set to true, the KDB kernel debugger does not print warning messages when it ignores certain breakpoints. If the no_brkpt_warning value is set to false, the KDB kernel debugger prints warning messages when it ignores certain breakpoints.

Aliases
setup

Example
The following is an example of how to use the set subcommand:

KDB(0)> set
No toggle name current value

1 no_symbol false
2 mst_wanted true
3 screen_size 24
4 power_pc_syntax true
5 origin 00000000
6 unix_symbols_start_from 00001000
7 hexadecimal_wanted true
8 screen_previous false
9 display_stack_frames false
10 display_stacked_regs false
11 64_bit true
12 ldr_segs_wanted false
13 emacs_window false
14 local_breakpoint_attach thread
15 kdb_stop_all_cpu true
17 kext_IF_active true
18 trace_back_lookup false
19 IPI_enable true
20 scroll false
21 edit noedit
24 no_brkpt_warning false
25 default_xmalloc_heap 00000000
KDB(0)> dc waitproc 5
   .waitproc+000000 mflr r0
   .waitproc+000004 mfcr r12
   .waitproc+000008 std r31,FFFFFFF8(stkp)
   .waitproc+00000C std r30,FFFFFFF0(stkp)
   .waitproc+000010 std r29,FFFFFFE8(stkp)
KDB(0)> set origin 100
   5 origin 00000100
KDB(0)> dc waitproc 5
   .waitproc+000000 (ORG+00026CB8) mflr r0
   .waitproc+000004 (ORG+00026CBC) mfcr r12
   .waitproc+000008 (ORG+00026CC0) std r31,FFFFFFF8(stkp)
   .waitproc+00000C (ORG+00026CC4) std r30,FFFFFFF0(stkp)
   .waitproc+000010 (ORG+00026CC8) std r29,FFFFFFE8(stkp)
KDB(0)> set scroll false
   20 scroll false
dbgopt subcommand

Purpose
The `dbgopt` subcommand toggles low-level tracing options within the kernel.

Syntax
`dbgopt`

Parameters
The `dbgopt` subcommand presents a menu that allows the user to enable rc.boot tracing and tracing of exec calls. The tracing enabled by this subcommand is performed using the kernel `printf` function and is unrelated to the system trace facility.

Aliases
No aliases.

Example
The following is an example of how to use the `dbgopt` subcommand:

```
KDB(0)> dbgopt
Debug options:
--------------
1. Toggle rc.boot tracing - currently DISABLED
2. Toggle tracing of exec calls - currently DISABLED
q. Exit

Enter option: 2

Debug options:
--------------
1. Toggle rc.boot tracing - currently DISABLED
2. Toggle tracing of exec calls - currently ENABLED
q. Exit

Enter option: q

KDB(0)>
```
varset subcommand

Purpose
The varset subcommand creates a new user-defined variable.

Note: In the KDB kernel debugger, user variables are persistent across invocations of the debugger but not across system reboots. In the kdb command, user variables are not persistent across invocations.

Syntax
varset name [value]

Parameters
name Specifies the name of a user variable. If it does not already exist, the variable is created. Otherwise, the value of the existing variable is changed. Variable names are case sensitive and can consist of letters, numbers, and the underscore (_) character.
value Is a string assigned verbatim to the user variable specified by name. If omitted, the user variable is assigned an empty string. The value can contain spaces.

After a variable is created, any occurrence of the variable name in a subcommand is replaced with the value assigned to that variable.

If any variable substitutions occur, the resulting subcommand is printed between two less than and two greater than signs before it is run. For example, <<dw kdb_avail l>>.

All variable substitutions are done before any additional parsing of the subcommand, and the substitutions are done on a textual basis. This allows a single variable to expand into multiple subcommand parameters.

Aliases
No aliases.

Example
The following is an example of how to use the varset subcommand:
KDB(0)> varset myvar kdb_avail
KDB(0)> dw myvar
<<dw kdb_avail>>
kdb_avail+000000: 00000001 00000000 0800004C 00001C43 ...........L...C
KDB(0)> varset myvar kdb_avail 1
KDB(0)> dw myvar
<<dw kdb_avail 1>>
kdb_avail+000000: 00000001 ....
KDB(0)>
varlist subcommand

Purpose
The varlist subcommand displays all user-defined variables previously created with the varset subcommand.

Syntax
varlist

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the varlist subcommand:

```
KDB(0)> varset myvar kdb_avail
KDB(0)> varlist
Slot   Name       Value
  0   myvar      kdb_avail
KDB(0)>
```
varrm subcommand

Purpose
The varrm subcommand removes user-defined variables previously created with the varset subcommand.

Syntax
varrm name

Parameters
name Specifies the user variable to remove. Variable names are case sensitive and consist of letters, numbers, and the underscore (_) character.

Aliases
No aliases.

Example
The following is an example of how to use the varrm subcommand:

```
KDB(0)> varlist
Slot   Name   Value
0   myvar   kdb_avail
KDB(0)> varrm myvar
KDB(0)> varlist
Slot   Name   Value
```

KDB Kernel debugger and kdb command
his subcommand

Purpose
The his subcommand prints a history of user input. A parameter can be used to specify the number of historical entries to display.

Syntax
his [value]

Parameters

value Indicates a decimal value or expression indicating the number of previous user entries to display.

Each historical entry can be recalled and edited for use with the usual control characters (as in emacs).

Aliases
hi, hist

Example
No example.
**debug subcommand**

**Purpose**
The `debug` subcommand prints additional information while the KDB kernel debugger is running to help ensure that the debugger is functioning properly.

**Syntax**
`debug [options]`

**Parameters**
`options` Specifies the debug option to be turned on or off. View possible values by specifying the `?` flag.

If the `debug` subcommand is invoked with no parameters, the currently-active debug options are displayed.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `debug` subcommand:

```
kdb> debug ? //debug help
vmm HW lookup debug... on with arg 'dbg1++', off with arg 'dbg1--'
vmm tr/tv cmd debug... on with arg 'dbg2++', off with arg 'dbg2--'
vmm Sw lookup debug... on with arg 'dbg3++', off with arg 'dbg3--'
symbol lookup debug... on with arg 'dbg4++', off with arg 'dbg4--'
stack trace debug..... on with arg 'dbg5++', off with arg 'dbg5--'
BRKPT debug (list)... on with arg 'dbg61++', off with arg 'dbg61--'
BRKPT debug (instr)... on with arg 'dbg62++', off with arg 'dbg62--'
BRKPT debug (suspend)... on with arg 'dbg63++', off with arg 'dbg63--'
DABR debug (address)... on with arg 'dbg71++', off with arg 'dbg71--'
DABR debug (register)... on with arg 'dbg72++', off with arg 'dbg72--'
DABR debug (status)... on with arg 'dbg73++', off with arg 'dbg73--'
BRAT debug (address)... on with arg 'dbg81++', off with arg 'dbg81--'
BRAT debug (register)... on with arg 'dbg82++', off with arg 'dbg82--'
BRAT debug (status)... on with arg 'dbg83++', off with arg 'dbg83--'
BRKPT debug (context)... on //this debug feature is enabled
kdb> debug dbg5++ //enable debug mode
stack trace debug..... on
```

KDB(4)> f //stack frame in debug mode
thread+00100 STACK:
=== Look for traceback at 0x00015278
=== Got traceback at 0x00015280 (delta = 0x00000008)
=== has_tboff = 1, tb_off = 0x08
=== Trying to find Stack Update Code from 0x000151A8 to 0x00015278
=== Found 0x9421F9A0 at 0x000151B8
=== Trying to find Stack Restore Code from 0x000151A8 to 0x0001527C
=== Trying to find Registers Save Code from 0x000151A8 to 0x00015278
[00015278]//waitproc+0000000 ()
=== Look for traceback at 0x00015274
=== Got traceback at 0x00015280 (delta = 0x0000000C)
=== has_tboff = 1, tb_off = 0x08
[00015274]//waitproc+0000000 ()
=== Look for traceback at 0x0002F400
=== Got traceback at 0x0002F420 (delta = 0x00000020)
=== has_tboff = 1, tb_off = 0x30
// Invoke command from command line that calls open
Breakpoint 0024FDE8 stwu stk,FFFFFFB0(stkp) stkp=2FF3B3C0,FFFFFFB0(stkp)=2FF3B370
KDB(0)> time //Report time from leaving the debugger till the break
Command: time Aliases:
Elapsed time since last leaving the debugger:
2 seconds and 121211136 nanoseconds.
KDB(0)>
! subcommand

Purpose
The ! subcommand serves as a shell escape and provides a way to run UNIX commands without leaving the kdb command. This subcommand is only available in the kdb command.

Note: If output logging is enabled through the logfile and loglevel kdb command options, the output produced by the ! subcommand is not included in the log file.

Syntax
! [command]

Parameters

command
Passes a command verbatim to a newly spawned UNIX shell for running.

Aliases
No aliases.

Example
The following is an example of how to use the ! subcommand:

(0)> ! ls
... .dtprofile bin lib sbin
.: .mozilla dev lost+found tftpboot
.TAuthority .sh_history dfs lpp tmp
.Xauthority .wmrc etc mnt unix
.bash_history : gsa opt usr
.dbxhist TT_DB home proc var
.dt audit krb5 project
(0>
Chapter 9. Leaving kdb subcommands

The subcommands in this category are used to exit the kdb command and the KDB kernel debugger, shutdown the machine and reboot the machine. These subcommands include the following:

- v
- vreboot
- vhalt
e subcommand

Purpose
The e subcommand exits the kdb command and KDB kernel debugger.

Syntax
e [dump]

Parameters

dump
Indicates that a system dump will be created when you exit the KDB kernel debugger. The optional dump parameter is only applicable to the KDB kernel debugger. The dump argument can be specified to force an operating system dump. The method used to force a dump depends on how the KDB kernel debugger was invoked.

The KDB kernel debugger can be invoked in the following ways:

panic
If the KDB kernel debugger was invoked by the panic call, force the dump by typing q dump and pressing Enter. If another processor enters the KDB kernel debugger after that (for example, a spin-lock timeout), exit the KDB kernel debugger.

When the dump is complete, control is returned to the KDB kernel debugger and the LEDs show xxxx.

halt display
If the KDB kernel debugger was invoked by a halt display (C20 on the LED), type q and press Enter.

When the dump is complete, the LEDs show 888 102 700 0c0.

soft reset
If the debugger was invoked by a soft reset (that is, pressing the reset button once), complete the following:

1. Move the key on the server.
   If the key was in the SERVICE position at boot time, move it to the NORMAL position. Otherwise, move the key to the SERVICE position.
   Note: Forcing a dump using this method requires that you know what the key position was at boot time.

2. Type quit and press Enter.
   Do this once for each CPU.

break in
You cannot create a dump if the debugger was invoked with the break method (^). When the dump is in progress, _0c9 displays on the LEDs while the dump is copied to disk hd7 or disk hd6.

The e subcommand allows you to exit the KDB kernel debugger session and return to the system with all breakpoints installed in memory. To leave KDB kernel debugger without breakpoints, use the ca subcommand.

Aliases
q, g

Example
No example.
reboot subcommand

Purpose
The reboot subcommand reboots the machine. This subcommand issues a prompt for confirmation that a reboot is desired before beginning the reboot.

Note: This subcommand is only available within the KDB kernel debugger. It is not included in the kdb command.

Syntax
reboot

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the reboot subcommand:
KDB(0)> reboot //reboot the machine
Do you want to continue system reboot? (y/[n]):> y
Rebooting ...
halt subcommand

Purpose
The `halt` subcommand shuts down the machine.

Note: This subcommand is only available within the KDB kernel debugger. It is not included in the `kdb` command.

Syntax
`halt`

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the `halt` subcommand:
```
KDB(0)> halt
Halting...
```
Chapter 10. Changing context subcommands

The subcommands in this category are used to change the context that is being debugged. These subcommands include the following:

- sw
- cpu
- context
- runcpu
sw subcommand

Purpose
The *sw* subcommand allows a selected thread to be considered the current thread.

Syntax
```
sw [ {th_slot | th_Address} | {u | k} ]
```

Parameters
- **u**: Switches to user address space for the current thread.
- **k**: Switches to kernel address space for the current thread.
- **th_slot**: Specifies a thread slot number. This parameter must be a decimal value.
- **th_Address**: Specifies the address of a thread slot. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The *u* and *k* flags can be used to switch between the user and kernel address space for the current thread.

By default, KDB shows the virtual space for the current thread. Threads can be specified by slot number or address. The current thread can be reset to its initial context by entering the *sw* subcommand with no parameters. For the KDB kernel debugger, the initial context is also restored whenever you exit the KDB kernel debugger.

Aliases
- **switch**

Example
The following is an example of how to use the *sw* subcommand:

KDB(0)> sw 12 //switch to thread slot 12
Switch to thread: <thread+000900>
KDB(0)> f //print stack trace
thread+000900 STACK:
[000215FC]e_block_thread+000250 ()
[00021C48]e_sleep_thread+000070 (??, ??, ??)
[000200F4]errread+00009C (??, ??)
[001C89B4]rdevread+00009C (??, ??, ??, ??, ??)
[001C89B4]spec_rdwr+00008C (??, ??, ??, ??, ??, ??, ??, ??, ??, ??)
[001BD80C]rwuio+0000CC (??, ??, ??, ??, ??, ??)
[001BDF40]rdwr+000184 (??, ??, ??, ??)
[001BDD68]kreadv+000064 (??, ??, ??, ??, ??, ??)
[00037D8].sys_call+000000 ()
[00046B68].read+000028 (??, ??, ??, ??)
[1000167C].child+000120 ()
[10001A84].main+0000E4 (??, ??)
[1000014C].__start+00004C ()
KDB(0)> dr sr //display segment registers
s0 : 00000000 s1 : 007FFFFF s2 : 000000B7 s3 : 007FFFFF s4 : 007FFFFF
s5 : 007FFFFF s6 : 007FFFFF s7 : 007FFFFF s8 : 007FFFFF
s9 : 007FFFFF s10 : 007FFFFF s11 : 007FFFFF s12 : 007FFFFF s13 : 6000058B
s14 : 6000058B s15 : 6000058B
KDB(0)> sw u //switch to user context
KDB(0)> dr sr //display segment registers
s0 : 60000000 s1 : 600009B1 s2 : 60000AB7 s3 : 007FFFFF s4 : 007FFFFF
s5 : 007FFFFF s6 : 007FFFFF s7 : 007FFFFF s8 : 007FFFFF s9 : 007FFFFF
Now it is possible to look at user code
For example, find how read() is called by child()

KDB(0)> dc 1000167C //print child() code (seg 1 is now valid)
1000167C bl <1000A1BC>
KDB(0)> dc 1000A1BC 6 //print child() code
1000A1BC lwz r12,244(toc)
1000A1C0 stw toc,14(stkp)
1000A1C4 lwz r0,0(r12)
1000A1C8 lwz toc,4(r12)
1000A1CC mtctr r0
1000A1D0 bcctr
... //find stack pointer of child() routine with 'set 9; f'

The following example shows some of the differences between kernel and user mode for 64-bit process:

(0)> sw k //kernel mode
(0)> dr msr //kernel machine status register
msr : 00001080 bit set: ME IR DR
(0)> dr r1 //kernel stack pointer
r1 : 2FF2B850 2FF2B850
(0)> f //stack frame (kernel MST)

Chapter 10. Changing context subcommands
(0)> f //stack frame (kernel MST extension)
thread+002A98 STACK:
[0000000000000000000581D4]sleep+000000 (0000000000000064 [??])
[000000000000000000000478]main+0000CC (000000010000000001, 00000002000FEB78)
[00000000000000000000023C]__start+000044 ()
**cpu subcommand**

**Purpose**
The `cpu` subcommand allows you to switch from the current processor to the specified processor.

**Syntax**
```plaintext
cpu [ cpu number | any ]
```

**Parameters**
- `cpu number` Specifies the CPU number. This value must be a decimal value.
- `any` Unblocks switched processors.

Without a parameter, the `cpu` subcommand prints processor status.

For the `kdb` command, the processor status displays the address of the Per Processor Data Area (PPDA) for the processor, the current thread for the processor, and the Current Save state Address (CSA).

For the KDB kernel debugger, the processor status indicates the current state of the processor (for example, stopped, switched, debug, and so forth). A switched processor is blocked until the next `start` or `cpu` subcommand. Switching between processors does not change the processor state.

**Note:** If a selected processor cannot be reached, you can go back to the previous processor by typing `\tw\tw` twice.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `cpu` subcommand:

```
KDB(4)> cpu //display processors status
cpu 0 status VALID SWITCHED action SWITCH
cpu 1 status VALID SWITCHED action SWITCH
cpu 2 status VALID SWITCHED action SWITCH
cpu 3 status VALID SWITCHED action SWITCH
cpu 4 status VALID DEBUG action RESUME
cpu 5 status VALID SWITCHED action SWITCH
cpu 6 status VALID SWITCHED action SWITCH
cpu 7 status VALID SWITCHED action SWITCH
KDB(4)> cpu 7 //switch to processor 7
Debugger entered via keyboard.
.waitproc+0000B0 lbz r0,0(r30) r0=0,0(r30)=ppda+001400
KDB(7)> cpu //display processors status
cpu 0 status VALID SWITCHED action SWITCH
cpu 1 status VALID SWITCHED action SWITCH
cpu 2 status VALID SWITCHED action SWITCH
cpu 3 status VALID SWITCHED action SWITCH
cpu 4 status VALID SWITCHED action SWITCH
cpu 5 status VALID SWITCHED action SWITCH
cpu 6 status VALID SWITCHED action SWITCH
cpu 7 status VALID DEBUG
KDB(7)> 
```
**ctx subcommand**

**Purpose**

The **ctx** subcommand is used to switch between cpu contexts when viewing a system memory dump.

**Note:** This subcommand is only available within the **kdb** command. It cannot be used with the KDB kernel debugger.

**Syntax**

```plaintext
ctx [cpu number]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu number</td>
<td>decimal value or expression indicating a CPU number. If the CPU number is not given as a parameter, the initial context is restored.</td>
</tr>
</tbody>
</table>

**Note:** You can select KDB context to see more information through the stack trace subcommand. For example, you could see a complete stack of a kernel panic. However, KDB context is available only if the running kernel is booted with KDB kernel debugger.

**Aliases**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

The following is an example of how to use the **ctx** subcommand:

```
$ kdb dump unix //dump analysis
Preserving 628325 bytes of symbol table
First symbol sys_resource
Component Names:
  1) proc
  2) thrd
  3) errlg
  4) bos
  5) vmm
  6) bscsi
  7) scdisk
  8) lvm
  9) tty
 10) netstat
 11) lent_dd

PFT:
 id.................0007
 raddr.....0000000000100000 eaddr....0000000001000000
 size.............00800000 align...........00001000
 valid.1 ro.....0 holes..0 io.....0 seg....1 wimg...2

PVT:
 id.................0008
 raddr.....00000000004B8000 eaddr....000000000480000
 size.............000FFD60 align...........00001000
 valid.1 ro.....0 holes..0 io.....0 seg....1 wimg...2
Dump analysis on POWER_PC POWER_604 machine with 8 cpu(s)
Processing symbol table...
 ....................done
(0)> stat  //machine status
RS6K_SMP_MCA POWER_PC POWER_604 machine with 8 cpu(s)
 ........... SYSTEM STATUS
```
AIX 4.3
Starting physical processor #1 as logical #1... done.
Starting physical processor #2 as logical #2... done.
Starting physical processor #3 as logical #3... done.
Starting physical processor #4 as logical #4... done.
Starting physical processor #5 as logical #5... done.
Starting physical processor #6 as logical #6... done.
Starting physical processor #7 as logical #7... done.

assert(v_lookup(sid,pno) == -1)

......  SYSTEM MESSAGES

Machine State Save Area [2FF38400]

iar : 00027AEC msp : 00010800 cr : 22222222 lr : 00243E58
ctr : 00000000 xer : 00000000 mg : 00000000
r0 : 000A7E74 r1 : 2FF38220 r2 : 002EBC70 r3 : 00013350 r4 : 00000000
r5 : 00000100 r6 : 00009030 r7 : 2FF38400 r8 : 00000106 r9 : 00000000
r10 : 00243E58 r11 : 2FF38400 r12 : 00001080 r13 : 0001C80 r14 : 2FF2A88
r15 : 20022D88 r16 : 20006A98 r17 : 20033128 r18 : 00000000 r19 : 0008AD56
r20 : 002A6038 r21 : 0000006A r22 : 00000000 r23 : 0000FFFF r24 : 00000100
r25 : 00003262 r26 : 00000000 r27 : 002B8AE0 r28 : 002A9F70 r29 : 00000001
r30 : 00003350 r31 : 00013350
s0 : 00000000 s1 : 007FFFFF s2 : 0000064B s3 : 007FFFFF s4 : 007FFFFF
s5 : 007FFFFF s6 : 007FFFFF s7 : 007FFFFF s8 : 007FFFFF s9 : 007FFFFF
s10 : 007FFFFF s11 : 00010010 s12 : 00002002 s13 : 6001F01F s14 : 00040004
s15 : 007FFFFF
prev 00000000 kjmpbuf 00000000 stackfix 00000000 intpri 0B
curid 0008AD56 sralloc E01E0000 ioalloc 00000000 backt 00
flags 00 tid 00000000 excp_type 00000000
fpscr 00000000 fpeu 01 fpinfo 00 fpscrx 00000000
o_iar 00000000 o_toc 00000000 o_arg1 00000000
excbranch 00000000 o_vaddr 00000000 mstext 00000000
Except:
csr 00000000 dsisr 40000000 bit set: DSISR_PFT
srval 600008640 dar 2FF2FF8 dsirr 00000106

[00027AEC].backt+000000 (00013350, 00000000 [??])
[00243E54]vms_delete+0004DC (??)
[00256838]shmfreews+000080 ()
[000732B4]freeuspace+000010 ()
[00072EAC]kexitx+000688 (??)
(4)> ctx //AIX context of CPU 4
Restore initial context
(4)> f //current stack
thread+031920 STACK:
[00027AEC].backt+000000 (00013350, 00000000 [??])
[00243E54]vms_delete+0004DC (??)
[00256838]shmfreews+000080 ()
[000732B4]freeuspace+000010 ()
[00072EAC]kexitx+000688 (??)
(4)>
run cpu subcommand

Purpose
The run cpu subcommand allows you to run any other kdb subcommand to for every processor in the system. It is intended for use with subcommands such as the f subcommand for which the output depends on the current processor in the KDB kernel debugger.

Syntax
run cpu cmd

Parameters


The specified command only runs on processors that the KDB kernel debugger has stopped. If errors occur when the command is run on a particular processor, the run cpu subcommand continues and runs the command on the next processor. The run cpu subcommand can be stopped by pressing Ctrl+C.

Aliases
No aliases.

Example
The following is an example of how to use the run cpu subcommand:
KDB(0)> run cpu f

--- CPU #0 ---
pthread+000200 STACK:
[00026078]waitproc_find_run_queue+00018C (0000000000000001 [??])
[000285DC]waitproc+000134 ()
[000DE8F8]procentry+000010 (??, ??, ??, ??)

--- CPU #1 ---
pthread+000300 STACK:
[00026124]waitproc_find_run_queue+000238 (0000000000000080 [??])
[000285DC]waitproc+000134 ()
[000DE8F8]procentry+000010 (??, ??, ??, ??)
KDB(0)>
Chapter 11. Calculator and converter subcommands

The subcommands in this category are used to convert decimal numbers to other formats and evaluate decimal and hexadecimal expressions. These subcommands include the following:

- `hcal`
- `dcal`
- `conv`
**hcal and dcal subcommands**

**Purpose**
The *hcal* subcommand evaluates hexadecimal expressions and displays the result in both hexadecimal and decimal. The *dcal* subcommand evaluates decimal expressions and displays the result in both hexadecimal and decimal.

**Syntax**
- `hcal` *HexadecimalExpression*
- `dcal` *DecimalExpression*

**Parameters**
- *HexadecimalExpression* Specifies the hexadecimal expression to be evaluated.
- *DecimalExpression* Specifies the decimal expression to be evaluated.

**Aliases**
- *hcal* – cal
- *dcal* has no alias.

**Example**
The following is an example of how to use the *dcal* subcommand and the *hcal* subcommand:

```kdb
KDB(0)> hcal 0x10000 //convert a single value
Value hexa: 00010000
Value decimal: 65536
KDB(0)> dcal 1024*1024 //convert an expression
Value decimal: 1048576
Value hexa: 00100000
KDB(0)> set 11 //64 bits printing
64_bit is true
KDB(0)> hcal 0-1 //convert -1
Value hexa: FFFFFFFFFFFFFFFF
Value decimal: -1
Unsigned: 18446744073709551615
KDB(0)> set 11 //32 bits printing
64_bit is false
KDB(0)> hcal 0-1 //convert -1
Value hexa: FFFFFFFF
Value decimal: -1
Unsigned: 4294967295
```
conv subcommand

Purpose
The conv subcommand converts an arbitrary base number to a decimal, binary, octal, or hexadecimal number.

Syntax
conv [-b | -d | -o | -x | -a base] [-s] value

Parameters
-b  Specifies that the number to convert specified by the value parameter is a binary number.
-d  Specifies that the number to convert specified by the value parameter is a decimal number.
-o  Specifies that the number to convert specified by the value parameter is an octal number.
-x  Specifies that the number to convert specified by the value parameter is a hexadecimal number.
-a base  Specifies that the number to convert specified by the value parameter is a number with the arbitrary base of base. The number must be between 2 and 36 inclusive.
-s  Extends the left-most, one-bit sign of the number to convert specified by the value parameter.
value  Specifies the number to convert.

Aliases
No aliases.

Example
The following is an example of how to use the conv subcommand:

KDB(0)> conv 1101
Binary: 0000000000000000000000000000000000000000000000000000000000001101
Octal: 0000000000000000002115
Decimal: 1101
Hex: 000000000000044D
KDB(0)> conv -b 1101
Binary: 0000000000000000000000000000000000000000000000000000000000001101
Octal: 0000000000000000000015
Decimal: 13
Hex: 000000000000000D
KDB(0)> conv -b -s 1101
Binary: 1111111111111111111111111111111111111111111111111111111111111111
Octal: 17777777777777777
Decimal: -3
Hex: FFFFFFFF
KDB(0)>
Chapter 12. CPU start and stop subcommands

The subcommands in this category are used to selectively hold processors in kdb spin loops and then release them back to general operating system use. These subcommands include the following:

- start
- stop
**start and stop subcommands**

**Purpose**
The `start` subcommand starts all processors or a specific processor. The `stop` subcommand stops all processors or a specific processor.

**Note:** These subcommands are only available within the KDB kernel debugger. They are not included in the `kdb` command.

**Syntax**
- `start cpu_number | all`
- `stop cpu_number | all`

**Parameters**
- `cpu_number` Specifies the CPU number to start or stop. This parameter must be a decimal value.
- `all` Indicates that all processors are to be started or stopped.

When a processor is stopped, it is looping inside the KDB kernel debugger and the processor does not go back to the operating system.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `start` subcommand and the `stop` subcommand:

```
KDB(1)> stop 0 //stop processor 0
KDB(1)> cpu //display processors status
cpu 0 status VALID STOPPED action STOP
cpu 1 status VALID DEBUG
KDB(1)> start 0 //start processor 0
KDB(1)> cpu //display processors status
cpu 0 status VALID action START
cpu 1 status VALID DEBUG
KDB(1)> b sy_decint //set break point
KDB(1)> e //exit the debugger
Breakpoint .sy_decint+000000 mflr r0 <.dec_flih+000014>
KDB(0)> cpu //display processors status
cpu 0 status VALID DEBUG WAITING
cpu 1 status VALID DEBUG WAITING
KDB(0)> cpu 1 //switch to processor 1
KDB(0)> b sy_decint+000000 mflr r0 <.dec_flih+000014>
KDB(1)> cpu //display processors status
KDB(1)> cpu 0 //switch to processor 0
KDB(0)> cpu //display processors status
KDB(0)> q //exit the debugger
```

74  KDB Kernel debugger and kdb command
Chapter 13. Basic display subcommands

The subcommands in this category display stack frames, system statistics and information about processors. These subcommands include the following:

- \[\text{v}\]
- \[\text{v} \text{status}\]
- \[\text{v} \text{stat}\]
- \[\text{v} \text{pr}\]
- \[\text{v} \text{symptom}\]
**f subcommand**

**Purpose**
The `f` subcommand displays all of the stack frames from the current instruction as deep as possible. Interrupts and system calls are crossed and the user stack is displayed.

**Syntax**
```
f [+x | -x] [th {slot | address}]
```

**Parameters**

+**x**
  Includes hexadecimal addresses as well as symbolic names for calls on the stack. This option remains set for future invocations of the stack subcommand until it is changed using the -x flag.

-**x**
  Suppresses the display of hexadecimal addresses for functions on the stack. This option remains in effect for future invocations of the stack subcommand until it is changed using the +x flag.

**slot**
  Indicates the thread slot number. It is a decimal value.

**Address**
  Indicates the effective address for a thread slot. It is a hexadecimal address, hexadecimal expression, or symbol.

In the user space, trace back allows the display of symbolic names, but the KDB kernel debugger cannot directly access these symbols. Use the +x toggle to have hexadecimal addresses displayed (for example, to put a break point on one of these addresses). If invoked with no parameter, the stack for the current thread is displayed. The stack for a particular thread can be displayed by specifying its slot number or address.

**Note:** The amount of data displayed can be controlled through the `mst_wanted` and `display_stack_wanted` options of the `set` subcommand. For more information, see "set subcommand" on page 44.

For some compilation options, specifically -O, routine parameters are not saved in the stack. KDB warns about this by displaying [??] at the end of the line. In this case, the displayed routine parameters might be wrong.

**Aliases**

stack, where

**Example**
The following is an example of how to use the `f` subcommand. In the following example, a break point is set on `v_gettlock` and when the break point is encountered, the stack is displayed. The first parameter of the `open()` syscall is displayed and saved by `copen()` in register R31. Register R31 is saved in the stack by `openpath()`. The first parameter is found by looking at the memory pointed to by register R31.

```
KDB(2)> f //show the stack
thread=012540 STACK:
[00004C84]v_gettlock+000000 (00012049, C0011E80, 00000080, 00000000 [??]) <-- Optimized code, note [??] 
[000051C8]v_pregettlock+0000B4 (??, ??, ??, ??) 
[00012E8B]sync_vcs1+0000D8 (??, ??) 
[00012049].backt+000000 (00012049, C0011E80 [??]) <-- Optimized code, note [??] 
[00012049]vm_gettlock+000020 (??, ??) 
[0019A64C]write+00013C (??) 
[0019D194]comlist+0001CC (??, ??) 
[0019D4F0]_commit+000030 (00000000, 00000001, 09C6E9E8, 399028AA, 0000A46F, 0000E2AA, 2D3A4EAA, 2FF3A730) 
[001E1B18]jfs_setattr+000258 (??, ??, ??, ??, ??, ??, ??) 
```
In the following example, you must find what the lsfs subcommand is waiting for. The answer is given with getfssize parameters, which are saved in the stack.

```
# ps -ef|grep lsfs
root 63046 39258 0 Apr 01 pts/1 0:00 lsfs
# kdb
Preserving 587377 bytes of symbol table
First symbol sys_resource
PFT:
id..................0007
raddr..............01000000 eaddr........80000000

Chapter 13. Basic display subcommands
```
size.........01000000 align............01000000
valid .1 ros. 0 holes . 0 io ... 0 seg. 0 wimg ... 2

PVT:
id..................0008
raddr...............003BC000 eaddr...............82000000
size................001FFD00 align............00001000
valid .1 ros. 0 holes . 0 io ... 0 seg. 0 wimg ... 2
(0)> dcal 63046 //print hexadecimal value of PID
Value decimal: 63046
Value hexa: 000F646
(0)> tpid 0000F646 //show threads of this PID
SLOT NAME STATE TID PRI CPU FLAGS WCHAN

thread+025440 795 lsfs SLEEP 31B31 03C 000 00000004 057D0858C
(0)> sw 795 //set current context on this thread
Switch to thread: <thread+025440>
(0)> f //show the stack
thread+025440 STACK:
[0000205C0]e_block_thread+00250 ()
[00002081C]e_sleep_thread+00040 (??, ??, ??)
[0002AAA0]iowait+00004C ()
[0002B40C]bread+00000DC (??, ??)
[0002AFA0]readblk+00000AC (??, ??, ??, ??)
[001E9D6B]spec_rdwr+00007C (??, ??, ??, ??, ??, ??, ??)
[001A6328]vnp_rdwr+000070 (??, ??, ??, ??, ??, ??, ??)
[00198278]rwulo+0000CC (??, ??, ??, ??, ??, ??, ??)
[001986AC]rdwr+000184 (??, ??, ??, ??, ??, ??)
[001984D4]kready+000064 (??, ??, ??, ??)
[000037D8].sys_call+000000 ()
[00046A18]read+000028 (??, ??, ??)
[1000A0E4]get_superblk+000054 (??, ??, ??)
[100035F8]read_super+000024 (??, ??, ??, ??, ??)
[10005C00]getfsblk+000040 (??, ??)
[10002D18].start+00004C ()

(0)> sw u //enable user context of the thread
(0)> dc 10005C00-a6 8 //look for parameters R3, R4, R5
10005B60 mflr r0
10005B64 stw r31,FFFFFFFC(stkp)
10005B68 stw r0,8(stkp)
10005B6C stwu stkp,FFFFFFE0(stkp)
10005B70 stw r3,108(stkp)
10005B74 stw r4,104(stkp)
10005B78 stw r5,10C(stkp)
10005B7C addi r3,r4,0
(0)> set 9 //print stack frame
display_stack_frames is true
(0)> f //show the stack
thread+025440 STACK:
[0000205C0]e_block_thread+00250 ()
...
[100035F8]read_super+000024 (??, ??, ??, ??, ??, ??)

=======================================================================
2FF225D0+104 //print parameters (offset 0x104 0x108 0x10c)
2FF22640: 20000C80 20000C78 00000000 00000004
(0)> d 20000C78 20 //print first parameter
20000C78: 2F74 6D70 2F73 7472 6970 655F 6673 2E32 /tmp/stripe_fs.2
20000C80: 3433 3632 0000 0000 0000 0000 0000 0004 4362...........

KDB Kernel debugger and kdb command
(0)> d 2000DCC8 20 //print second parameter
2000DCC8: 2F64 6576 2F73 6C76 3234 3336 3200 0000 /dev/slv24362...
2000DCDB: 0000 0000 0000 0000 0000 0000 0000 0004 .................
(0)> q //leave debugger
#
status subcommand

Purpose
The `status` subcommand displays information about what is currently running on each processor.

Syntax
```status [cpu]```

Parameters
- `cpu` Specifies the CPU number.

If no argument is specified, information is displayed for all processors.

Aliases
No aliases.

Example
The following is an example of how to use the `status` subcommand:

```
KDB(0)> status
CPU TID TSLOT PID PSLOT PROC_NAME
 0 205 2 204 2 wait
 1 307 3 306 3 wait
KDB(0)> status 1
CPU TID TSLOT PID PSLOT PROC_NAME
 1 307 3 306 3 wait
```
stat subcommand

Purpose
The stat subcommand displays system statistics that include the last kernel printf() messages still in memory.

Syntax
stat

Parameters
No parameters.

The following information is displayed for a processor that has crashed:
- Processor logical number
- Current Save Area (CSA) address
- LED value

For the KDB kernel debugger, this subcommand also displays the reason why the debugger was entered. There is one reason per processor.

Aliases
No aliases.

Example
The following is an example of how to use the stat subcommand:

KDB(6)> stat //machine status using the KDB kernel debugger
RS6K_SMP_MCA POWER_PC POWER_604 machine with 8 cpu(s)
SYSTEM STATUS:
sysname: AIX
nodename: jumbo32
release: 2
version: 4
machine: 00920312A000
nid: 920312A0
Illegal Trap Instruction Interrupt in Kernel
age of system: 1 day, 5 hr., 59 min., 50 sec.

SYSTEM MESSAGES

AIX 4.2
Starting physical processor #1 as logical #1... done.
Starting physical processor #2 as logical #2... done.
Starting physical processor #3 as logical #3... done.
Starting physical processor #4 as logical #4... done.
Starting physical processor #5 as logical #5... done.
Starting physical processor #6 as logical #6... done.
Starting physical processor #7 as logical #7... done.
<- end of buffer
CPU 6 CSA 00427EB0 at time of crash, error code for LEDs: 70000000

(0)> stat //machine status using the kdb command running on the dump file
RS6K_SMP_MCA POWER_PC POWER_604 machine with 4 cpu(s)
............ SYSTEM STATUS
sysname... AIX nodename.. zoo22
release... 3 version... 4
machine... 00909903A6 nid....... 989903A6
time of crash: Sat Jul 12 12:34:32 1997
age of system: 1 day, 2 hr., 3 min., 49 sec.

........ SYSTEM MESSAGES

AIX 4.3
Starting physical processor #1 as logical #1... done.
Starting physical processor #2 as logical #2... done.
Starting physical processor #3 as logical #3... done.
<- end_of_buffer

............. CPU 0 CSA 004ADEB0 at time of crash, error code for LEDs: 30000000
thread=01B438 STACK:
[00057F64]v_sync+0000E4 (B01C876C, 0000001F [?])
[000A4F0A]v_presync+000050 (?, ??)
[0002B05C]begbt_603_patch_2+000008 (?, ??)

Machine State Save Area [2FF3B400]
iar : 0002AF4C msr : 00001080 cr : 24224220 lr : 0023D474
ctr : 00000004 xer : 20000000 mq : 00000000
r0 : 00044F50 r1 : 2FF3A600 r2 : 002E62B8 r3 : 00000000 r4 : 07D17B60
r5 : E60B1438 r6 : 00025225 r7 : 00025225 r8 : 00000106 r9 : 00000004
r10 : 0023D474 r11 : 0023D470 r12 : 002894B8 r13 : 0028899C r14 : 0023A468
r15 : 0023A468 r16 : DEADBEEF r17 : DEADBEEF r18 : DEADBEEF r19 : 00000000
r20 : 0048D4C0 r21 : 0048D3E0 r22 : 07D6EE90 r23 : 00000140 r24 : 07D61360
r25 : 00025225 r26 : 00025225 r27 : 00025225 r28 : 00025225 r29 : 00025225
r30 : 07017B60 r31 : 07C76000
s0 : 00000000 s1 : 007FFFFF s2 : 00000109 s3 : 007FFFFF s4 : 007FFFFF
s5 : 007FFFFF s6 : 007FFFFF s7 : 007FFFFF s8 : 007FFFFF s9 : 007FFFFF
s10 : 007FFFFF s11 : 00000101 s12 : 00000135 s13 : 00000CC5 s14 : 00000404
s15 : 6000096E
prev 00000000 kjmbuf 2FF3A700 stackfix 00000000 intpri 0B
curid 00003C60 sralloc E01E0000 ialloc 00000000 backt 00
flags 00 tid 00000000 excp_type 00000000
fpscr 00000000 fpeu 00 fpinfo 00 fpscrx 00000000
o_iar 00000000 o_toc 00000000 o_arg1 00000000
excbranch 00000000 o_vaddr 00000000 mstext 00000000
Except:
csr 00000000 dsisr 40000000 bit set: DSISR_PFT
sval 00000000 dar 07CA705C dsirr 00000106

[0002AF4C].backt+000000 (00000000, 07D17B60 [?])
[0023A470]ilogsnc+00014C (?)
[002894B8]ilogsnc+000090 (?)
[0028899C]logmvnc+000124 (??, ??, ??, ??)
[0023A680]logafter+000100 (??, ??, ??)
[0023A466]commit2+0001EC (?)
[0023BF50]finicom+0000BC (??, ??)
[0023CC2C]comlist+0001F0 (??, ??)
[0029391C]jfs_rename+000794 (??, ??, ??, ??, ??)
[00248220]vnop_rename+000338 (??, ??, ??, ??, ??, ??)
[0026A168]rename+000380 (??, ??)
(0)
pr subcommand

Purpose
The pr subcommand displays memory as if it were of a specified type (C data structure).

Syntax
pr [type] address
pr -l offset \name [-e end_val] [type] address
pr -a count [type] address
pr -d default_type
pr -p pattern

Parameters
-  -l Displays data following a linked list. The pr subcommand follows the linked list until the value in the linked list pointer equals the ending value. The ending value is zero, unless it is changed with the -e parameter.
-  -e Changes the ending value used when you are displaying a linked list.
-  -a Displays the data as if it were an array whose elements are of the specified type.
-  -d Sets the default type.
-  default_type Indicates the type (C data structure) for which you want to display information. After you set the default type by using the -d parameter, it is the only type for which information is displayed.
-  -p Displays the defined symbols that match a specified pattern.
   type Specifies the type used to display the data.
   address Specifies the effective address of the data to be displayed.
   offset Specifies the offset of the linked list pointer in the data structure.
   name Specifies the name of the linked list pointer in the data structure.
   end_val Specifies the new ending value.
   count Specifies the number of elements to display.
   pattern Specifies the pattern.

Before a type can be used, it must be loaded into the kernel with the bosdebug -l command. The bosdebug command must be issued outside of kdb as the root user. It is not necessary to reboot the machine after running the bosdebug command.

Aliases
print

Example
The following is an example of how to use the pr subcommand:
KDB(0)> pr integer 3000 //use 'pr' without loading symbols
type definition not found

//Run the following as 'root' to load the symbols in intr.h into the kernel
# echo "#include <sys/intr.h>" >sym.c //symbol file to load into kernel
# echo "main() {}" >>sym.c
# cc -g -o sym.sym.c -qdbxextra //for 32-bit kernel
# cc -g -q64 -o sym.sym.c -qdbxextra //for 64-bit kernel
# bosdebug -l sym (load symbols into kernel)
Symbol table initialized. Loaded 297 symbols.
KDB(0)> pr integer 3000  //print data at 0x3000 as an integer
integer foo[0] = 0x4C696365;
KDB(0)> intr 19  //show interrupt handler table, slot 19
SLT INTRADDR HANDLER TYPE LEVEL PRI0 BID FLAGS
i_data+0004C  19  30047A80 00000000 0004 00000001 0000 900100C0 0040
i_data+0004C  19  0200C360 0200A908 0004 00000003 0000 900100C0 0040
KDB(0)> intr 30047A80  //show interrupt handler information at 0x30047A80
addr........... 30047A80 handler........ 00000000
bid............ 900100C0 bus_type........ 00000004 BID
next........... 0200C360 flags........... 00000040 LEVEL
level........... 00000001 priority........ 00000000 INTMAX
i_count........ 00000000
KDB(0)> pr intr 30047A80  //print this data as an 'intr' structure
struct intr {
    struct intr *next = 0x0200C360;
    int (*handler)() = 0x00000000;
    unsigned short bus_type = 0x0004;
    unsigned short flags = 0x0040;
    int level = 0x00000001;
    int priority = 0x00000000;
    ulong32int64_t bid = 0x900100C0;
    unsigned long i_count = 0x00000000;
} foo[0];
KDB(0)> pr 30047A80  //print data using default type
char foo[0] = 0x02 '';
KDB(0)> pr -d intr  //change default type to 'intr' structure
KDB(0)> pr 30047A80  //print data using new default type
struct intr {
    struct intr *next = 0x0200C360;
    int (*handler)() = 0x00000000;
    unsigned short bus_type = 0x0004;
    unsigned short flags = 0x0040;
    int level = 0x00000001;
    int priority = 0x00000000;
    ulong32int64_t bid = 0x900100C0;
    unsigned long i_count = 0x00000000;
} foo[0];
KDB(0)> pr -l next intr 30047A80  //print following the 'next' pointer
struct intr {
    struct intr *next = 0x0200C360;
    int (*handler)() = 0x00000000;
    unsigned short bus_type = 0x0004;
    unsigned short flags = 0x0040;
    int level = 0x00000001;
    int priority = 0x00000000;
    ulong32int64_t bid = 0x900100C0;
    unsigned long i_count = 0x00000000;
} foo;
struct intr {
    struct intr *next = 0x319A9020;
    int (*handler)() = 0x0200A908;
    unsigned short bus_type = 0x0004;
    unsigned short flags = 0x0040;
    int level = 0x00000003;
    int priority = 0x00000000;
    ulong32int64_t bid = 0x900100C0;
    unsigned long i_count = 0x00000000;
} foo;
struct intr {
    struct intr *next = 0x00000000;
    int (*handler)() = 0x02041AB8;
    unsigned short bus_type = 0x0004;
    unsigned short flags = 0x0040;
    int level = 0x00000003;
}
Chapter 13. Basic display subcommands
Flushed out all the symbols.

KDB(0)> pr integer 3000  //print after symbols unloaded

type definition not found
symptom subcommand

Purpose
The symptom subcommand displays the symptom string for a dump.

Note: The symptom subcommand is only available in the kdb command.

Syntax
symptom [-e]

Parameters
-e Writes the symptom string and the stack trace to the system errlog. The symptom string is displayed on the standard output.

If no parameters are used, the symptom subcommand displays the symptom string on the standard output.

The symptom subcommand is not valid on a running system. The optional -e flag creates an error log entry that contains the symptom string. This flag is normally only used by the system and not entered manually. The symptom string can be used to identify duplicate problems.

Aliases
No aliases.

Example
• The following example demonstrates the symptom command running on a dump:

```shell
<0> symptom
PIDS/5765C3403 LVLS/430 PCSS/PII MS/300 FLDS/uiocopyin VALU/7ce621ae
FLDS/uiomove VALU/13c
```

• The following example demonstrates the symptom subcommand with the -e flag running on a dump:

```shell
<0> symptom -e
PIDS/5765C3403 LVLS/430 PCSS/PII MS/300 FLDS/uiocopyin VALU/7ce621ae
FLDS/uiomove VALU/13c
```

• The corresponding system errlog entry is similar to the following:

```
LABEL: SYSDUMP_SYMP
....
Detail Data
DUMP STATUS
LED:300
csa:2ff3b400
uiocopyin_ppc 1c4
uiomove 13c
....
```
Chapter 14. Memory register display and decode subcommands

The subcommands in this category are used to display and decode the memory register. These subcommands include the following:

- `d`
- `dw`
- `dd`
- `dp`
- `dpw`
- `dpd`
- `dc`
- `dpc`
- `di`
- `dr`
- `ddvb`
- `ddvh`
- `ddvw`
- `dvd`
- `ddpb`
- `ddph`
- `ddpw`
- `ddpd`
d, dw, dd, dp, dpw, and dpd subcommands

Purpose
The d (display bytes), dw (display words), and dd (display double words) subcommands dump memory areas starting at a specified effective address. Access is done in real mode.

The dp (display bytes), dpw (display words), and dpd (display double words) subcommands dump memory areas starting at a specified real address.

Syntax
\[ d \text{ symbol} \mid \text{EffectiveAddress} \[ \text{count} \]\n\[ dw \text{ symbol} \mid \text{EffectiveAddress} \[ \text{count} \]\n\[ dd \text{ symbol} \mid \text{EffectiveAddress} \[ \text{count} \]\n\[ dp \text{ symbol} \mid \text{PhysicalAddress} \[ \text{count} \]\n\[ dpw \text{ symbol} \mid \text{PhysicalAddress} \[ \text{count} \]\n\[ dpd \text{ symbol} \mid \text{PhysicalAddress} \[ \text{count} \]\n
Parameters

\text{EffectiveAddress} \hspace{1cm} \text{Specifies the virtual (effective) address of the area to be dumped when the d, dw, or dd subcommands are used. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.}

\text{PhysicalAddress} \hspace{1cm} \text{Specifies the physical address of the area to be dumped when the dp, dpw or dpd subcommands are used. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.}

\text{count} \hspace{1cm} \text{Specifies the number of bytes, words, or double words to display. This is a hexadecimal value. The number of bytes are displayed if the d subcommand or the dp subcommand are used. The number of words are displayed if the dw or dpw subcommand are used. The number of double words is displayed if the dd subcommand or the dpd subcommand are used. If no count is specified, 16 bytes of data are displayed.}

Any of the display subcommands can be continued from the last address displayed by using the Enter key.

Aliases
\[ d \rightarrow \text{dump} \]

Example
The following is an example of how to use the d, dw, dd, dp, dpw, and dpd subcommands:

KDB(0)> d utname //display data at utname
utname+000000: 4149 5800 0000 0000 0000 0000 0000 0000 AIX..............
KDB(0)> d utname 8 //display 8 bytes of data at utname
utname+000000: 4149 5800 0000 0000 AIX.....
KDB(0)> //’enter key’ to display the next 8 bytes of data
utname+000000: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 AIX..............
KDB(0)> dw utname 8 //display 8 words of data at utname
utname+000000: 41495800 00000000 00000000 00000000 AIX..............
KDB(0)> dp utname+000000: 00000000 00000000 00000000 00000000 ..............
KDB(0)> dpw utname+000000: 0000000000000000 0000000000000000 AIX..............
KDB(0)> dpd utname+000000: 0000000000000000 0000000000000000 ..............
utsname+000020: 3030303030303030 4130303000000000 00000000A000....
utsname+000030: 0000000000000000 0000000000000000 0000000000000000 0000000000000000
KDB(0)> tr utsumer //find physical address of utsmmer
Physical Address = 000000003D2860
KDB(0)> dp 3D2860 //display data using physical address
003D2860: 4149 5800 0000 0000 0000 0000 0000 0000 0000 AIX............
KDB(0)> dpw 3D2860 //display data as words using physical address
003D2860: 41495800 00000000 00000000 00000000 AIX............
KDB(0)> dpd 3D2860 //display data as double-words using physical address
003D2860: 4149580000000000 0000000000000000 AIX............
KDB(0)>
dc and dpc subcommands

Purpose
The dc and dpc subcommands decode instructions.

Syntax

```
dc effectiveaddress [count]
dpc physicaladdress [count]
```

Parameters

- `effectiveaddress`: Specifies the effective or virtual address of the code to disassemble. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.
- `physicaladdress`: Specifies the physical or real address of the code to disassemble. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.
- `count`: Indicates the number of instructions to be disassembled. The value specified must be a decimal value or decimal expression.

Aliases

- `dc` – `dis`
- `dpc` has no aliases.

Example

The following is an example of how to use the dc and the dpc subcommands:

```
KDB(0)> set 4
power_pc_syntax is true
KDB(0)> dc resume_pc 10 //prints 10 instructions
.resume_pc+000000 1b 1r0,3454(0) 3454=Trconflag
.resume_pc+000004 mfspr 1r15,0
.resume_pc+000008 cmpi cr0,r0,0
.resume_pc+00000c 1wz toc,4208(0) toc=TOC,4208=g_toc
.resume_pc+000010 1wz r30,4C(r15)
.resume_pc+000014 1wz r14,40(r15)
.resume_pc+000018 1wz r31,8(r30)
.resume_pc+00001c bne cr0.eq,<.resume_pc+0001BC>
.resume_pc+000020 1ha r28,2(r30)
.resume_pc+000024 1wz r29,0(r14)
```

```
KDB(0)> dc mttb 5 //prints mttb function
.mttb+000000 1i 1r0,0
.mttb+000004 mttbl X r0 //X shows that these instructions
.mttb+000008 mttbu X r3 //are not supported by the current architecture
.mttb+00000C mttbl X r4 //POWER PC 601 processor
.mttb+000010 blr
```

```
KDB(0)> set 4 //set toggle for POWER family RS syntax
power_pc_syntax is false
KDB(0)> dc resume_pc 10 //prints 10 instructions
.resume_pc+000000 1b 1r0,3454(0) 3454=Trconflag
.resume_pc+000004 mfspr 1r15,110
.resume_pc+000008 cmpi cr0,r0,0
.resume_pc+00000c 1toc,4208(0) toc=TOC,4208=g_toc
.resume_pc+000010 1r30,4C(r15)
.resume_pc+000014 1r14,40(r15)
.resume_pc+000018 1r31,8(r30)
.resume_pc+00001c bne cr0.eq,<.resume_pc+0001BC>
.resume_pc+000020 1ha r28,2(r30)
.resume_pc+000024 1r29,0(r14)
```
KDB(4)> dc scdisk_pm_handler
    scdisk_pm_handler+000000  stmw  r26,FFFFFFE8(stkp)
KDB(4)> tr scdisk_pm_handler
Physical Address = 1D7CA1C0
KDB(4)> dpc 1D7CA1C0
1D7CA1C0  stmw  r26,FFFFFFE8(stkp)
di subcommand

Purpose
The di subcommand decodes the given hexadecimal instruction word.

Syntax
\texttt{di hexadecimal\_instruction}

Parameters

\textit{hexadecimal\_instruction} \hspace{1cm} Specifies the hexadecimal instruction word to be decoded.

The hexadecimal instruction word displays the actual instruction, with the operations code and the operands, of the given hexadecimal instruction. The \texttt{di} subcommand accepts a user input hexadecimal instruction word and decodes it into the actual instruction word in the form of the operations code and the operands.

Aliases
decode

Example
The following is an example of how to use the \texttt{di} subcommand:

\begin{verbatim}
KDB(0)> di 7Ce6212e
stwx     r7,r6,r4
KDB(0)>
\end{verbatim}
**dr subcommand**

**Purpose**
The `dr` subcommand displays general purpose, segment, special, or floating point registers.

**Syntax**
```
dr [gp | sr | sp | fp | vmx | reg_name]
```

**Parameters**
- **gp** Displays general purpose registers.
- **sr** Displays segment registers.
- **sp** Displays special purpose registers.
- **fp** Displays floating point registers.
- **vmx** Displays the current contents of vector registers. This is not the contents of the currently running thread’s vector register state unless the thread is the current owner of the vector unit.
- **reg_name** Displays a specific register by name.

The current thread context is used to locate the values to display. The `sw` subcommand can be used to change the context to other threads.

If no parameter is given, the general purpose registers are displayed.

For BAT registers, the `dbat` and `ibat` subcommands must be used.

**Aliases**
No aliases.

**Example**
1. The following is an example of how to use the `dr` subcommand:

```
KDB(0)> dr?
Usage: dr [sp|sr|gp|fp|hmt|vmx|<reg.name>]
Usage: mr [sp|sr|gp|fp]<reg.name>
```

```
sp reg. name: iar msp cr lr ctr xer mq asr
.............. dsisr dar dec sdr0 sdr1 srr0 srr1 dabr
.............. dabrx rtcu rct1 tubi tbl sprg0 sprg1 sprg2
.............. sprg3 pvr ear fpecr ctrl hid0 hid1
.............. hid4 hid5 ladr dmiss iaccess dcmp icmp hash1
.............. hash2 rpa buscsr 12cr 12sr imc sia sda
.............. imru imrl mmcr0 mmcr1 pmc1 pmc2 pmc3
.............. pmc4 pmc5 pmc6 pmc7 pmc8
sr reg. name: s0 s1 s2 s3 s4 s5 s6 s7
.............. s8 s9 s10 s11 s12 s13 s14 s15
gp reg. name: r0 r1 r2 r3 r4 r5 r6 r7
.............. r8 r9 r10 r11 r12 r13 r14 r15
.............. r16 r17 r18 r19 r20 r21 r22 r23
.............. r24 r25 r26 r27 r28 r29 r30 r31
fp reg. name: f0 f1 f2 f3 f4 f5 f6 f7
.............. f8 f9 f10 f11 f12 f13 f14 f15
.............. f16 f17 f18 f19 f20 f21 f22 f23
.............. f24 f25 f26 f27 f28 f29 f30 f31
.............. fpscr
vmx reg. name: vr0 vr1 vr2 vr3 vr4 vr5 vr6 vr7
.............. vr8 vr9 vr10 vr11 vr12 vr13 vr14 vr15
.............. vr16 vr17 vr18 vr19 vr20 vr21 vr22 vr23
.............. vr24 vr25 vr26 vr27 vr28 vr29 vr30 vr31
.............. vscr vrsave
```
hmt reg. name: rctr1 thctl thto dormiar dormmsr
KDB(0) > dr //print general purpose registers
r0 : 00003730 r1 : 2FEDFF88 r2 : 00211B6C r3 : 00000000 r4 : 00000003
r5 : 007FFFFF r6 : 002F9300 r7 : 2FEAFFFC r8 : 00000009 r9 : 20019CCB
r10 : 00000008 r11 : 0004B840 r12 : 009B7000 r13 : 2003FC60 r14 : DEADBEFE
r15 : 00000000 r16 : DEADBEEF r17 : 2003F028 r18 : 00000000 r19 : 2000916B
r20 : 2003FD38 r21 : 2FEAFF3C r22 : 00000001 r23 : 2003F700 r24 : 2FE9E2E0
r25 : 2FE0000 r26 : 00005454 r27 : E820B486 r28 : E3000E00 r29 : E6000BC0
r30 : 00353A6C r31 : 0000051I
KDB(0) > dr sp //print special registers
iar : 10001C48 msr : 0000F030 cr : 2B202B84 lr : 1000AF18
ctr : 1001DA04 xer : 00000003 mq : 00000DF4
disr : 42000000 dar : 394A8000 dec : 007DCC00
sdr1 : 00380007 srr0 : 10001C48 srr1 : 00000F30
dabr : 00000000 rtcu : 20C0E664 rtcl : 2E99E00
spr0 : 00A57440 sprq : 00000000 sprq2 : 00000000 sprq3 : 00000000
pid : 00000000 fpescr : 00000000 ear : 00000000 pvr : 00010001
hid0 : 8101FBC1 hid1 : 00004000 iabr : 00000000
KDB(0) > dr sr //print segment registers
s0 : 60000000 s1 : 6001377 s2 : 6001BDE s3 : 6001B7D s4 : 600143D
s5 : 6001F03D s6 : 60005C9 s7 : 007FFFFF s8 : 007FFFFF s9 : 007FFFFF
s10 : 007FFFFF s11 : 007FFFFF s12 : 007FFFFF s13 : 6000000A s14 : 007FFFFF
s15 : 60010102
KDB(0) > dr fp //print floating point registers
f0 : C027C28F5C28F5C3 f1 : 0003333339999999A f2 : 3FE3333333333333
f3 : 3FC9999999999999 f4 : 7F00000000000000 f5 : 0010000000000000
f6 : 4000000000000000 f7 : 000000009A068000 f8 : 7FF8000000000000
f9 : 00000000BA411000 f10 : 0000000000000000 f11 : 0000000000000000
f12 : 0000000000000000 f13 : 0000000000000000 f14 : 0000000000000000
f15 : 0000000000000000 f16 : 0000000000000000 f17 : 0000000000000000
f18 : 0000000000000000 f19 : 0000000000000000 f20 : 0000000000000000
f21 : 0000000000000000 f22 : 0000000000000000 f23 : 0000000000000000
f24 : 0000000000000000 f25 : 0000000000000000 f26 : 0000000000000000
f27 : 0000000000000000 f28 : 0000000000000000 f29 : 0000000000000000
f30 : 0000000000000000 f31 : 0000000000000000 fpscr : BA411000
KDB(0) > dr ctr //print CTR register
ctr : 1000ADA4
1000ADA4 cmpi cr0,r3,E7 r3=2FEAB008
KDB(0) > dr msr print MSR register
msr : 0000F030 bit set : EE PR FP ME IR DR
KDB(0) > dr cr
cr : 2B202B84 bits set in CR0 : EQ
----------------------------------------.CR1 : LT
----------------------------------------.CR2 : EQ
----------------------------------------.CR4 : EQ
----------------------------------------.CR5 : LT
----------------------------------------.CR6 : LT
----------------------------------------.CR7 : GT
KDB(0) > dr xer //print XER register
xer : 00000003 comparison byte : 0 length : 3
KDB(0) > dr iar //print IAR register
iar : 10001C48
10001C48 stw r12,4(stkp) r12=2B202B84,4(stkp)=2FEAFAFD4
KDB(0) > set 11 //enable 64 bits display on 620 machine
64 bit is true
KDB(0) > dr //display 620 general purpose registers
r0 : 0000000000244CF0 r1 : 0000000000259EB4 r2 : 000000000025A110
r3 : 000000000004A860 r4 : 0000000000000001 r5 : 0000000000000000
r6 : 0000000000000000 r7 : 0000000000010900 r8 : 000000000018DA00
r9 : 000000000015A820 r10 : 000000000018D000 r11 : 0000000000000000
r12 : 000000000023F05C r13 : 0000000000000000 r14 : 0000000000000000
r15 : 0000000000000000 r16 : 0000000000000000 r17 : 00000000003000F0
r18 : 0000000000000000 r19 : 0000000000000000 r20 : 0000000000225A48
r21 : 00000000001FF3E00 r22 : 00000000002259D0 r23 : 000000000025A12C
r24 : 0000000000000001 r25 : 0000000000000001 r26 : 00000000001FF4E0
r27 : 0000000000000001 r28 : 00000000001FF4A64 r29 : 00000000001FF4000
r30 : 000000000000034C r31 : 00000000001FF4A64

96 KDB Kernel debugger and kdb command
2. The following is an example of how to use the **dr** subcommand on a PCI machine to print one word at physical address 80000cfc::

```plaintext
KDB(0)> dr sp display 620 special registers
iar : 000000000023F288 msr : 0000000000021080 cr : 42000440
lr : 0000000000245738 ctr : 0000000000000000 xer : 00000000
mq : 00000000 asr : 0000000000000000
dsisr : 42000000 dar : 0000000000000000 dec : C3528E2F
sdrl : 01EC0000 srr0 : 000000000023F288 srrl : 0000000000021080
dabr : 0000000000000000 tbu : 00000002 tbl : AF33287B
sprg0 : 0000000000AAC00 sprg1 : 0000000000000040
sprg2 : 0000000000000000 sprg3 : 0000000000000000
plr : 0000000000000000 ear : 00000000 pvr : 00140201
hid0 : 7001C080 iabr : 0000000000000000
buscsr : 00000000008DC800 12cr : 0000000000000421A 12sr : 0000000000000000
mmcr0 : 00000000 pmc1 : 00000000 pmc2 : 00000000
sia : 0000000000000000 sda : 0000000000000000
KDB(0)>
```

```plaintext
2. The following is an example of how to use the **dr** subcommand on a PCI machine to print one word at physical address 80000cfc::

```
ddvb, ddvh, ddvw, ddvd, ddpb, ddph, ddpw, and ddpd subcommand

Purpose
The ddvb, ddvh, ddvw and ddvd subcommands can be used to access memory in translated mode, using an effective address already mapped. On a 64-bit machine, double words correctly aligned are accessed in a single load (ld) instruction with the ddvd subcommand.

The ddpb, ddph, ddpw and ddpd subcommands can be used to access memory in translated mode, using a physical address that will be mapped. On a 64-bit machine, double words correctly aligned are accessed in a single load (ld) instruction with the ddpd subcommand. The DBAT interface is used to translate this address in cache-inhibited mode.

Note: These subcommands are only available within the KDB kernel debugger. They are not included in the kdb command.

Syntax
ddvb EffectiveAddress [count]

ddvh EffectiveAddress [count]

ddvw EffectiveAddress [count]

ddvd EffectiveAddress [count]

ddpb PhysicalAddress [count]

ddph PhysicalAddress [count]

ddpw PhysicalAddress [count]

ddpd PhysicalAddress [count]

Parameters

EffectiveAddress Specifies the effective or virtual address of the starting memory area to display. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

PhysicalAddress Specifies the physical or real address of the starting memory area to display. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

count Specifies the number of bytes for ddvb and ddpb to display, specifies the number of half words for ddvh and ddph to display, specifies the number of words for ddvw and ddpw to display and specifies the number of double words for ddvd and ddpd to display. The count argument is a hexadecimal value.

I/O space memory (Direct Store Segment (T=1)) cannot be accessed when translation is disabled. The areas mapped by the bat command areas must also be accessed with translation enabled. Otherwise, cache controls are ignored.

Note: The subcommands that use effective addresses assume that mapping to real addresses is currently valid. No check is done by the KDB kernel debugger. The subcommands that use real addresses can be used to let KDB kernel debugger perform the mapping (attach and detach).
**Aliases**

The alias for:

- `ddvb` is `diob`
- `ddvh` is `diov`
- `ddvw` is `diow`
- `ddvd` is `diod`

There are no aliases for the following:

- `ddpb`
- `ddph`
- `ddpw`
- `ddpd`

**Example**

**Note:** The PowerPC 601 RISC Microprocessor is only available on AIX 5.1 and earlier.

The following is an example on the PowerPC 601 RISC Microprocessor:

```
KDB(0)> tr fff19610 //show current mapping
BAT mapping for FFF19610
  bepi 7FEO brpn 7FEO b1 001F v 1 wim 3 ks 1 kp 0 pp 2 s 0
  eaddr = FFC00000, paddr = FFC00000 size = 4096 KBytes
KDB(0)> ddvb fff19610 10 //print 10 bytes using data relocate mode enable
FFF19610: 0041 96B0 6666 CEEA 0041 A0B0 0041 AAB0 .A..ff...A...A..
KDB(0)> ddvw fff19610 4 //print 4 words using data relocate mode enable
FFF19610: 004196B0 76763346 0041A0B0 0041AAB0
KDB(0)>
```

The following is an example on a PCI machine:

```
KDB(0)> ddpw 80000cfc //print one word at physical address 80000cfc
80000CFC: D0000000 //Read is done in relocated mode, cache inhibited
KDB(0)>
```
Chapter 15. Memory search and extract subcommands

The subcommands in this category are used to search and extract information from memory. These subcommands include the following:

- `find`
- `findp`
- `ext`
- `extp`
find and findp subcommands

Purpose
The find and findp subcommands search for a specific pattern in memory.

Syntax
find [-s string]
find effectiveaddress pattern [mask | delta]
findp [-s string]
findp physicaladdress pattern [mask | delta]

Parameters
-s Indicates the pattern to be searched for is an ASCII string.
EffectiveAddress Specifies the effective or virtual address. Symbols, hexadecimal values, or hexadecimal
expressions can be used to specify the address.
PhysicalAddress Specifies the physical or real address. Symbols, hexadecimal values, or hexadecimal expressions
can be used to specify the address.
string Specifies the ASCII string to search for if the -s option is specified. The period (.) is used to
match any character.
pattern Specifies the hexadecimal value of the pattern to search for. The pattern is limited to one word in
length.
mask If a pattern is specified, a mask can be specified to eliminate bits from consideration for matching
purposes. This parameter is a one-word hexadecimal value.
delta Specifies the increment to move forward after an unsuccessful match. This parameter is a
one-word hexadecimal value.

The pattern that is searched for can either be an ASCII string, if the -s option is used, or a one word
hexadecimal value. If the search is for an ASCII string, the period (.) can be used to match any character.

A mask parameter can be used if the search is for a hexadecimal value. The mask is used to eliminate
bits from consideration. When it is checking for matches, the value from memory is ended with the mask
and then compared to the specified pattern for matching. For example, a mask of 7fffffff indicates that
the high bit is not to be considered. If the specified pattern was 0000000d and the mask was 7fffffff, the
values 0000000d and 8000000d are both considered matches.

A parameter can also be specified to indicate the delta that is applied to determine the next address to
check for a match. This ensures that the matching pattern occurs on specific boundaries. For example, if
you want to find the 0f0000ff pattern aligned on a 64-byte boundary, the following subcommand could be
used:
find 0f0000ff ffffffff 40

The default delta is one byte for matching strings and one word for matching a specified hexadecimal
pattern.

If the find or findp subcommands find the specified pattern, the data and address are displayed. Continue
the search from that point by pressing the Enter key.

Aliases
No aliases.
Example

The following is an example of how to use the **find** and the **findp** subcommands:

```
KDB(0)> tpid //print current thread
SLOT NAME   STATE   TID   PRI   CPUID   CPU   FLAGS   WCHAN
thread+002F40 63*nfsd RUN 03F8F 03C 000 00000000
KDB(0)> find lock_pinned 03F8F 00ffffff //search TID in the lock area
//compare only 24 low bits, on cache aligned addresses (delta 0x20)
lock_pinned+00D760: 00003F8F 00000000 0000005 00000000
KDB(0)> <CR/LF> //repeat last command
Invalid address E800F000, skip to (^C to interrupt)
...............
E8800000
Invalid address E8840000, skip to (^C to interrupt)
............... E9000000
Invalid address E9012000, skip to (^C to interrupt)
............... F0000000
KDB(0)> findp 0 E819D200 //search in physical memory
00F97C7C: E819D200 00000000 00000000 00000000
KDB(0)> <CR/LF> //repeat last command
05C4FB18: E819D200 00000000 00000000 00000000
KDB(0)> <CR/LF> //repeat last command
0F7550F0: E819D200 00000000 E60009C0 00000000
KDB(0)> <CR/LF> //repeat last command
0F927EE8: E819D200 00000000 05E62D28 00000000
KDB(0)> <CR/LF> //repeat last command
kdb_get_real_memory: Out of range address 1FFFFFFF
KDB(0)> find -s 01A86260 pse //search "pse" in pse text code
01A86C04: 7073 655F 6B64 6200 8062 0518 8063 0000 pse_kdb...b...c...
KDB(0)> <CR/LF> //repeat last command
01A92952: 7073 6562 7566 6361 6C6C 735F 696E 6974 psebufcalls_init
KDB(0)> <CR/LF> //repeat last command
01A939AE: 7073 655F 7265 766F 6B65 BEA1 FFD4 7D80 pse_revoke.....
KDB(0)> <CR/LF> //repeat last command
01A9456A: 7073 655F 7265 766F 6B65 6E65 6374 BF81 pse_bufcall.....
KDB(0)> <CR/LF> //repeat last command
01A94F5A: 7073 655F 7265 766F 6B65 6E65 6374 6E65 BEA1 FF04 7080 pse_revoke....).p
KDB(0)> <CR/LF> //repeat last command
01A9547E: 7073 655F 7365 6C65 7374 BE41 FFCB 7080 pse_select.A...}
KDB(0)> find -s 01A86260 pse......thread //how to use ".".
01A9586E: 7073 655F 626C 6F63 6B65 7468 7265 6164 pse_block_thread
KDB(0)> <CR/LF> //repeat last command
01A9F6EA: 7073 655F 736C 6565 705F 7468 7265 6164 pse_sleep_thread
```

Chapter 15. Memory search and extract subcommands 103
ext and extp subcommands

Purpose
The ext and extp subcommands display a specific area from a structure.

Syntax
ext [-p] EffectiveAddress delta [size | count]
extp [-p] PhysicalAddress delta [size | count]

Parameters

- **-p** Indicates that the delta argument is the offset to a pointer to the next area.
  
  **EffectiveAddress** Specifies the effective or virtual address at which to begin displaying values. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
  
  **PhysicalAddress** Specifies the physical or real address at which to begin displaying values. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
  
  **delta** Specifies the offset to the next area to be displayed or the offset from the beginning of the current area to a pointer to the next area. This argument is a hexadecimal value.
  
  **size** Specifies the offset to the next area to be displayed or the offset from the beginning of the current area to a pointer to the next area. This argument is a hexadecimal value.
  
  **count** Specifies the hexadecimal value that indicates the number of words to display.

If the -p flag is not specified, these subcommands display the number of words indicated in the size argument. They then increment the address by the delta and display the data at that location. This procedure is repeated for the number of times indicated in the count parameter.

If the -p flag is specified, these subcommands display the number of words indicated by the size parameter. The next address from which data is to be displayed is then determined by using the value at the current address plus the offset indicated in the delta parameter (for example, *(addr+delta)). This procedure is repeated for the number of times indicated in the count parameter.

If an array exists, it can be traversed displaying the specified area for each entry of the array. These subcommands can also be used to traverse a linked list displaying the specified area for each entry.

Aliases
No aliases.

Example
The following is an example of how to use the exp and the expt subcommands:

```
KDB(0)> ppda
KDB Kernel debugger and kdb command
```
ppda_pal[3].................00000000
phy_cpuid....................0000
sradid.......................0000
pvpa.........................0000000001130400
slb_reload...................0000
slb_index....................0000
slb_stoimask.................0000
slb_stoibits...............0000
slb_g_start................0000000000000000
slb_g_nesids..............0000000000000000
slb_ksp_start...........0000000000000000
slb_ksp_nesids...........0000000000000000
slb_glp_start...........0000000000000000
slb_glp_nesids.........0000000000000000
slb_glp_tbl.............0000000000000000
slb_lgpg_start.........0000000000000000
slb_lgpg_nesids........0000000000000000
slb_slbsave..............0000000000000000
slb_recurse_cnt.............0000
slb_stab_addr..........0000000000000000

KDB(0)> ext -p 000000000184EE00 0 10 2 // csa address from the ppda
mststack+020E00: F0000000 2FF47600 00000000 00000000.../v........
mststack+020E10: 00000000 00000000 00000000 00000000 .............
mststack+020E20: 00000000 00000000 00000000 00000000 .............
mststack+020E30: 00000000 000302A0 00000000 0003023C ...........
__ublock+000000: 00000000 00000000 00000000 00000000 .............
__ublock+000010: 00000000 00000000 00000000 00000000 .............
__ublock+000020: 00000000 00000000 00000000 00000000 .............
__ublock+000030: 00000000 00251380 00000000 0028828......Q8\\

KDB(0)> ext 000000000184EE00 3000 10 2 // mstsave address from the ppda
mststack+01DE00: 00000000 0184EE00 00000000 00000000 .............
mststack+01DE10: 00000000 00000000 00000000 00000000 .............
mststack+01DE20: 00000000 00000000 00000000 00000000 .............
mststack+01DE30: 00000000 00000000 00000000 00000000 .............
mststack+020E00: F0000000 2FF47600 00000000 00000000.../v........
mststack+020E10: 00000000 00000000 00000000 00000000 .............
mststack+020E20: 00000000 000302A0 00000000 0003023C ...........

KDB(0)>
Chapter 16. Memory modification subcommands

The subcommands in this category are used to modify memory. These subcommands include the following:

- m
- mw
- md
- mpw
- mpd
- st
- stc
- sth
- mdvb
- mdvh
- mdvw
- mdvd
- mdpb
- mdph
- mdpw
- mdpd
- mr
m, mw, md, mp, mpw, and mpd subcommands

Purpose
The m (modify bytes), mw (modify words) and md (modify double words) subcommands modify memory starting at a specified effective address. The mp (modify bytes), mpw (modify words) and mpd (modify double words) subcommands modify memory starting at a specified real address.

These subcommands are only available within the KDB kernel debugger. They are not included in the kdb command.

Syntax
m effectiveaddress
mw effectiveaddress
md effectiveaddress
mp physicaladdress
mpw physicaladdress
mpd physicaladdress

Parameters
effectiveaddress Specifies the effective or virtual address of the starting memory area to modify. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
physicaladdress Specifies the physical or real address of the starting memory area to modify. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

Read or write access can be in virtual or real mode.

These subcommands are interactive. Each modification is entered one-by-one. The first unexpected input stops modification. For example, a period (.) can be used to indicate the end of the data. If a break point is set at the same address, use the mw subcommand to maintain break point coherency.

Note: Symbolic expressions are not allowed as input.

Aliases
No aliases.

Example
The following is an example of how to use the mw and msubcommands to do a patch:

```
KDB(0)> dc @iar //print current instruction
.open+000000  mf1r r0
KDB(0)> mw @iar //nop current instruction
.open+000000:  7C0802A6  = 60000000
.open+000004:  93E1FFFC  = . //end of input
KDB(0)> dc @iar //print current instruction
.open+000000  ori r0,r0,0
KDB(0)> m @iar //restore current instruction byte per byte
.open+000000:  60  = 7C
.open+000001:  00  = 00
.open+000002:  00  = 02
```
KDB(0)> dc @iar //print current instruction
KDB(0)> tr @iar //physical address of current instruction
Physical Address = 001C5BA0
KDB(0)> mwp 001C5BA0 //modify with physical address
KDB(0)> dc @iar 5 //print instructions
KDB(0)> mw open+c //restore instruction
KDB(0)> dc open+c //print instruction
st, stc, and sth subcommands

Purpose
The st, stc and sth subcommands store data at a specified address.

Syntax

st EffectiveAddress Value

stc EffectiveAddress Value

sth EffectiveAddress Value

Parameters

EffectiveAddress Specifies the effective address to which the data will be stored. Hexadecimal values or hexadecimal expressions can be used in specification of the address.

Value Specifies the data value to be stored. The value stored is:
- One word if you use the st subcommand
- One character if you use the stc subcommand
- One half-word if you use the sth subcommand

Aliases
No aliases.

Example
The following is an example of how to use the st, the stc and the sth subcommands:

KDB(0)> dw 20
00000020: 00000000 00000000 00000000 00000000

KDB(0)> st 20 11111111
KDB(0)> dw 20
00000020: 11111111 00000000 00000000 00000000

KDB(0)> st 20 2
KDB(0)> dw 20
00000020: 00000002 00000000 00000000 00000000

KDB(0)> st 20 0
KDB(0)> dw 20
00000020: 00000000 00000000 00000000 00000000

KDB(0)> stc 20 33
KDB(0)> dw 20
00000020: 33000000 00000000 00000000 00000000

KDB(0)> st 20 0
KDB(0)> dw 20
00000020: 00000000 00000000 00000000 00000000

KDB(0)> sth 20 4444
KDB(0)> dw 20
00000020: 44440000 00000000 00000000 00000000

KDB(0)> st 20 0
KDB(0)> dw 20
00000020: 00000000 00000000 00000000 00000000

KDB(0)> dw 20
00000020: 00000000 00000000 00000000 00000000
mdvb, mdvh, mdvw, mdvd, mdpb, mdph, mdpw, mdpd subcommands

Purpose

The mdvb, mdvh, mdvw, and mdvd subcommands can be used to access memory in translated mode, using an effective address already mapped. On a 64-bit machine, double words are accessed by the mdvd subcommand in a single store instruction.

The mdpb, mdph, mdpw, and mdpd subcommands access memory in translated mode, using a physical address that will be mapped. On a 64-bit machine, correctly-aligned double words are accessed by the mdpd subcommand in a single store instruction. The DBAT interface is used to translate this address in cache-inhibited mode.

Note: These subcommands are only available within the KDB kernel debugger. They are not included in the kdb command.

Syntax

mdvb effectiveaddress
mdvh effectiveaddress
mdvw effectiveaddress
mdvd effectiveaddress
mdpb physicaladdress
mdph physicaladdress
mdpw physicaladdress
mdpd physicaladdress

Parameters

effectiveaddress

Specifies the virtual (effective) address of the memory to modify. It can be symbols, hexadecimal values, or hexadecimal expressions.

physicaladdress

Specifies the real (physical) address of the memory to modify. It can be symbols, hexadecimal values, or hexadecimal expressions.

These subcommands are available to write in I/O space memory.

To avoid bad effects, memory is not read before, only the specified write is performed with translation enabled. Access can be in bytes, half words, words or double words.

Note: The subcommands using effective addresses assume that mapping to real addresses is currently valid. No check is done by KDB kernel debugger. The subcommands using real addresses allow KDB kernel debugger to do the mapping (attach and detach).

Aliases

The aliases are:

mdvb – miob
mdvh – miob
mdvw – miow
There are no aliases for the following:

- mdpb
- mdph
- mdpw
- mdpd

**Example**

The following is an example on the PowerPC 601 RISC Microprocessor:

**Note:** The PowerPC 601 RISC Microprocessor is only supported on AIX 5.1 and earlier.

```
KDB(0)> tr FFF19610 //print physical mapping
BAT mapping for FFF19610
  bepi 7FE0 brpn 7FE0 b1 001F v 1 wim 3 ks 1 kp 0 pp 2 s 0
eaddr = FFC00000, paddr = FFC00000 size = 4096 KBytes
KDB(0)> mdvb fff19610 //byte modify with data relocate enable
FFF19610: ?? = 00
FFF19611: ?? = 00
FFF19612: ?? = . //end of input
KDB(0)> mdvw fff19610 //word modify with data relocate enable
FFF19610: ??????? = 004196B0
FFF19614: ??????? = . //end of input
KDB(0)>
```

The following is an example on a PCI machine:

```
KDB(0)> mdpw 80000cf8 //change one word at physical address 80000cf8
80000CF8: ??????? = 84000080
80000CFC: ??????? = . //Write is done in relocated mode, cache inhibited
KDB(0)> ddpw 80000cfc //print one word at physical address 80000cfc
80000CFC: D2000000
KDB(0)> mdpw 80000cfc //change one word at physical address 80000cfc
80000CFC: ??????? = 00000000
80000CFC: ??????? = .
KDB(0)> mdpw 80000cf8 //change one word at physical address 80000cf8
80000CF8: ??????? = 8c000080
80000CFC: ??????? = .
KDB(0)> ddpw 80000cfc //print one word at physical address 80000cfc
80000CFC: D2000080
```
**mr subcommand**

**Purpose**
The *mr* subcommand modifies general purpose, segment, special, or floating point registers.

**Syntax**
```
mr [gp | sr | sp | fp | reg_name]
```

**Parameters**
- **gp**: Modifies general purpose registers.
- **sr**: Modifies segment registers.
- **sp**: Modifies special purpose registers.
- **fp**: Modifies floating point registers.
- **reg_name**: Modifies a specific register by name.

Individual registers can also be selected for modification by register name. The current thread context is used to locate the register values to be modified. Use the *sw* subcommand to change the context to other threads. When the register being modified is in the *mst* subcommand context, the KDB kernel debugger alters the Machine Save State Area. When the register being modified is a special register, the register is altered immediately. Symbolic expressions are allowed as input.

If the *gp*, *sr*, *sp*, or *fp* options are used, modification of all of the registers in the group is allowed. The current value for a single register is shown and modification is allowed. Then, the value for the next register is displayed for modification. Entry of an invalid character, such as a period (.), ends modification of the registers. If the value for a register is to be left unmodified, press Enter to continue to the next register for modification.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the *mr* subcommand:
```
KDB(0)> dc @iar //print current instruction
.open+000000  mflr  r0
KDB(0)> mr iar //modify current instruction address
iar : 001C5BA0 = @iar+4
KDB(0)> dc @iar //print current instruction
.open+000004  stw  r31,FFFFFFFC(stkp)
KDB(0)> mr iar //restore current instruction address
iar : 001C5BA4 = @iar-4
KDB(0)> dc @iar //print current instruction
.open+000000  mflr  r0
KDB(0)> mr sr //modify first invalid segment register
s0 : 00000000 = <CR/LF>
s1 : 60000323 = <CR/LF>
s2 : 20001E1E = <CR/LF>
s3 : 007FFFFF = 0
s4 : 007FFFFF = . //end of input
KDB(0)> dr s3 //print segment register 3
s3 : 00000000
KDB(0)> mr s3 //restore segment register 3
s3 : 00000000 = 007FFFFF
KDB(0)> mr f29 //modify floating point register f29
f29 : 0000000000000000 = 000333335999999A
KDB(1)> mr vr0 //modify vector register vr0
```
vr0: 00000000000000000000000000000000 = 1122334455667788 <CR/LF>
    = 99aabccddeeff00
KDB(0)> dr f29
f29: 000333335999999A
KDB(1)> dr vr0 //dump vector register vr0
vr0: 112233445566778899AABBCCDDEEFF00
KDB(0)> u
Uthread [2FF3B400]:
     save@......2FF3B400  fpr@......2FF3B550
...
KDB(0)> dd 2FF3B550 20
__ublock+000150:  C027C2BF5C28F5C3  000333335999999A  '.\{....33Y...
__ublock+000160:  3FE33333333333  3FC9999999999999  ?.333333?....
__ublock+000170:  7FF0000000000000  01000000C0000000  ...............
__ublock+000180:  7FF0000000000000  000000009A068000  0................
__ublock+000190:  7FF0000000000000  000000009A068000  0................
__ublock+0001A0:  0000000000000000  0000000000000000  ...............
__ublock+0001B0:  0000000000000000  0000000000000000  ...............
__ublock+0001C0:  0000000000000000  0000000000000000  ...............
__ublock+0001D0:  0000000000000000  0000000000000000  ...............
__ublock+0001E0:  0000000000000000  0000000000000000  ...............
__ublock+0001F0:  0000000000000000  0000000000000000  ...............
__ublock+000200:  0000000000000000  0000000000000000  ...............
__ublock+000210:  0000000000000000  0000000000000000  ...............
__ublock+000220:  0000000000000000  0000000000000000  ...............
__ublock+000230:  0000000000000000  000333335999999A  ........33Y...
__ublock+000240:  0000000000000000  0000000000000000  ...............
KDB(0)>

Note: The vr0 register modifies the current vector register contents. The vector register state of the current thread is not modified unless the thread is the current owner of the vector unit. The 16-byte vector input is entered as 8 bytes followed by a carriage return and then followed by 8 bytes.
Chapter 17. Breakpoint and steps subcommands

The subcommands in this category are used to set and clear breakpoints and provide step functions. These subcommands include the following:

- \texttt{b}
- \texttt{lb}
- \texttt{ca}
- \texttt{r}
- \texttt{gt}
- \texttt{n}
- \texttt{s}
- \texttt{B}
b subcommand

Purpose
The b subcommand sets a permanent global breakpoint in the code. KDB kernel debugger checks whether a valid instruction is trapped.

Note: This subcommand is only available within the KDB kernel debugger. It is not included in the kdb command.

Syntax
b [-p | -v] [ address]

Parameters
- -p Indicates that the breakpoint address is a physical or real address.
- -v Indicates that the breakpoint address is a effective or virtual address.
- address Specifies the address of the breakpoint. This may either be a physical address or a virtual address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

If an invalid instruction is detected, a warning message is displayed. If the warning message is displayed, the breakpoint should be removed; otherwise, memory can be corrupted.

Aliases
brk

Example
The following example is before VMM setup:
KDB(0)> b vsi //set break point on vsi()
.vsi+000000 (real address:002AA5A4) permanent & global
KDB(0)> e //exit debugger
...
Breakpoint
.vsi+000000 stk r29,FFFFFFF4(stkp) <.mainstk+001EFC> r29=isync_sc1+000040,FFFFFFF4(stkp)=.mainstk+001EFC

The following example is after VMM setup:
KDB(0)> b //display current active break points
No breakpoints are set.
KDB(0)> b 0 //set break point at address 0
WARNING: break point at 00000000 on invalid instruction (00000000) 00000000 (sid:00000000) permanent & global
KDB(0)> c 0 //remove break point at address 0
KDB(0)> b vmvcs //set break point on vmvcs()
.vmvcs+000000 (sid:00000000) permanent & global
KDB(0)> b i_disable //set break point on i_disable()
.i_disable+000000 (sid:00000000) permanent & global
KDB(0)> e //exit debugger
...
Breakpoint
.i_disable+000000 mfmsr r7 <start+001008> r7=DEADBEEF
KDB(0)> b //display current active break points
0: .vmvcs+000000 (sid:00000000) permanent & global
1: .i_disable+000000 (sid:00000000) permanent & global
KDB(0)> c 1 //remove break point slot 1
KDB(0)> b //display current active break points
0: .vmvcs+000000 (sid:00000000) permanent & global
KDB(0)> e //exit debugger
... Breakpoint
.vmvc8+000000 mfr r10 <.initcom+000120>
KDB(0)> ca //remove all break points
**lb subcommand**

**Purpose**
The `lb` subcommand sets a permanent local breakpoint in the code for a specific context.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the `kdb` command.

**Syntax**

```
lb [-p | -v] [address]
```

**Parameters**

- `-p` Indicates that the breakpoint address is a physical or real address.
- `-v` Indicates that the breakpoint address is an effective or virtual address.
- `address` Specifies the address of the breakpoint. This can be either an effective or physical address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The context can either be CPU-based or thread-based. Either context is controllable through a `set subcommand` on page 44 option. Each `lb` subcommand associates one context with the local breakpoint and up to eight different contexts can be set for each local breakpoint. The context is the effective address of the current thread entry in the thread table or the current processor number.

If the `lb` subcommand is used with no parameters, all current trace and breakpoints are displayed.

If an address is specified, the break is set with the context of the current thread or CPU. To set a break using a context other than the current thread or CPU, change the current context using the `sw subcommand` on page 60 and the `cpu subcommand` on page 63.

If a local breakpoint is hit with a context that was not specified, a message is displayed, but a break does not occur.

By default, KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical address or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address. After VMM is setup, the `-p` parameter must be used to set a breakpoint in real-mode for code that is not mapped V=R. Otherwise, the KDB kernel debugger expects a virtual address and translates the address.

**Aliases**

`lbrk`

**Example**
The following is an example of how to use the `lb` subcommand:

```
KDB(0)> b execv //set break point on execv()
Assumed to be [External data]: 001F4200 execve
Ambiguous: [Ext func]
001F4200 .execve
.execve+000000 (sid:00000000) permanent & global
KDB(0)> e //exit debugger
... Breakpoint
.execve+000000  mflr r0  <.svc_flih+00011C>
KDB(0)> ppda //print current processor data area
```
Per Processor Data Area [00086E40]

csa......................2FEE0000  mstack......................0037CDB0
tpowner....................00000000  curthread..............E60008C0
...
KDB(0)> lb kexit  //set local break point on kexit()
.kexit+000000 (sid:00000000) permanent & local < ctx: thread+0008C0 >
KDB(0)> b  //display current active break points
0:  .execve+000000 (sid:00000000) permanent & global
1:  .kexit+000000 (sid:00000000) permanent & local < ctx: thread+0008C0 >
KDB(0)> e  //exit debugger
...
Warning, breakpoint ignored (context mismatched):
.kexit+000000  mflr  r0  <._exit+000020>
Breakpoint
.kexit+000000  mflr  r0  <._exit+000020>
KDB(0)> ppda  //print current processor data area

Per Processor Data Area [00086E40]

csa......................2FEE0000  mstack......................0037CDB0
tpowner....................00000000  curthread..............E60008C0
...
KDB(0)> lc 1 thread+0008C0  //remove local break point slot 1
c, lc, and ca subcommands

Purpose
The c, lc and ca subcommands clear breakpoints.

Note: This subcommand is only available within the KDB kernel debugger. It is not included in the kdb command.

Syntax
\[ c \{ slot | -p | -v \} \text{Address} \]

\[ ca \]

\[ lc \{ slot | -p | -v \} \text{Address} [ctx] \]

Parameters

- **-p** Indicates that the breakpoint address is a physical or real address.
- **-v** Indicates that the breakpoint address is an effective or virtual address.
- **slot** Specifies the slot number of the breakpoint. This parameter must be a decimal value.
- **Address** Specifies the address of the breakpoint. This may either be a physical or virtual address. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
- **ctx** Specifies the context to be cleared for a local break. The context may either be a CPU or thread specification.

The ca subcommand erases all breakpoints. The c and lc subcommands erase only the specified breakpoint. The c subcommand clears all contexts for a specified breakpoint. The lc subcommand can be used to clear a single context for a breakpoint. If a specific context is not specified, the current context is used to determine which local breakpoint context to remove.

By default, the KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address.

Note: Slot numbers are not fixed. To clear slot 1 and slot 2 type c 2; c 1 or type c 1; c 1. Do not enter c 1; c 2.

Aliases

- c – cl
- lc – lcl

Example
The following is an example of how to use the c and the ca subcommands:

\[ KDB(1)\> b //list breakpoints \]
\[ 0: \text{.halt\_display+000000 (sid:00000000) permanent & global} \]
\[ 1: \text{.v\_exception+000000 (sid:00000000) permanent & global} \]
\[ 2: \text{.v\_loghalt+000000 (sid:00000000) permanent & global} \]
\[ 3: \text{.p\_slih+000000 (sid:00000000) trace (hit: 0)} \]

\[ KDB(1)\> c 2 //clear breakpoint slot 2 \]
\[ 0: \text{.halt\_display+000000 (sid:00000000) permanent & global} \]
\[ 1: \text{.v\_exception+000000 (sid:00000000) permanent & global} \]
\[ 2: \text{.p\_slih+000000 (sid:00000000) trace (hit: 0)} \]

\[ KDB(1)\> c v\_exception //clear breakpoint set on v\_exception \]
0: .halt_display+000000 (sid:00000000) permanent & global
1: .p_slih+000000 (sid:00000000) trace {hit: 0}
KDB(1)> ca //clear all breakpoints
0: .p_slih+000000 (sid:00000000) trace {hit: 0}
r and gt subcommands

Purpose
The r and gt subcommands set non-permanent breakpoints. Non-permanent breakpoints are local breakpoints that are cleared after they are used.

Note: This subcommand is only available within the KDB kernel debugger. It is not included in the kdb command.

Syntax
r

gt [-p | -v] [address]

Parameters
-p Indicates that the breakpoint address is a physical or real address.
-v Indicates that the breakpoint address is an effective or virtual address.
address Specifies the address of the breakpoint. This may either be a physical or real address. Symbols, hexadecimal values, or hexadecimal expressions may be used in specification of the address.

The r subcommand sets a breakpoint on the address found in the lr register. In the SMP environment, it is possible to reach this breakpoint on another processor. For this reason, it is important to use the thread or process local break point.

The gt subcommand performs the same function as the r subcommand, but the address must be specified for the gt subcommand.

By default, the KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is physical. If the subcommand is entered after VMM initialization, the address is virtual (effective address). After VMM is initialized, the -p flag must be used to set a breakpoint in real-mode code that is not mapped V=R, otherwise KDB kernel debugger expects a virtual address and translates the address.

Aliases
r – return

gt has no aliases.

Example
The following is an example of how to use the r and the gt subcommands:

```
KDB(2)> b _iput  //enable break point on _iput()
._iput+000000 (sid:00000000) permanent & global
KDB(2)> e  //exit debugger
...
Breakpoint
._iput+000000  stmw r29,FFFFFFF4(stkp) <2FF3B1CC> r29=0A4C6C20,FFFFFFF4(stkp)=2FF3B1CC
KDB(6)> !
thread+014580 STACK:
[002263F4]_iput+000000 (0A4C6C20, 0571A808 [??])
[002263F4]jfs_rele+0000B4 (??)
[00226058]vnode_rele+000018 (??)
[00232178]vnode_close+000058 (??)
[002266C8]closef+0000C8 (??)
```

122  KDB Kernel debugger and kdb command
Chapter 17. Breakpoint and steps subcommands
n, s, S, and B subcommand

Purpose
The n, s, S and B subcommands provide step functions.

Note: These subcommands are only available within the KDB kernel debugger. They are not included in the kdb command.

Syntax
n [count]
s [count]
S [count]
B [count]

Parameters

- count Specifies the number of times the subcommand runs.
- n Runs the number of instructions specified by count, but it treats subroutine calls as a single instruction. If specified without a number, it runs one instruction.
- s Runs the number of instructions specified by the count parameter.
- S Runs instructions until it encounters a bl or br branch instruction. If the count parameter is used, the number specifies how many bl and br instructions are reached before the KDB Kernel Debugger stops.
- B Runs instructions until it encounters any branch instruction. If the count parameter is used, the number specifies how many branch instructions are reached before the KDB Kernel Debugger stops.

On POWER-based machines, steps are implemented with the SE bit of the msr status register of the processor. This bit is automatically associated with the thread or process context. The thread or process context can migrate from one processor to another.

You can interrupt any of these subcommands by pressing the Del key. Every time the KDB kernel debugger takes a step, it checks to see whether the Del key was pressed. This allows you to break into the KDB kernel debugger if the call is taking an inordinate amount of time.

If no intervening subcommands are run, any of the step commands can be repeated by pressing the Enter key.

Be aware that when you use these subcommands, an exception to the processor is made for each of the debugged program’s instruction. One side-effect of exceptions is that it breaks reservations. The stcwd instruction cannot succeed if any breakpoint occurred after the last larwx instruction. The net effect is that you cannot use these subcommands with lock and atomic routines. If you do, you loop in the lock routine.

Some instructions are broken by exceptions. For example, rfi moves to and from srr0 srr1. The KDB kernel debugger tries to prevent this by printing a warning message.

When you want to take control of a sleeping thread, switch to the sleeping thread with the sw subcommand and type the s subcommand. The step is set inside the thread context, and when the thread runs again, the step breakpoint occurs.
Aliases

The aliases are:

n – nexti
s – stepi

There are no aliases for the following:

S
B

Example

The following is an example of how to use the n, s, and B subcommands:

```
KDB(1)> b .vno_close+00005C //enable breakpoint on vno_close+00005C
KDB(1)> vno_close+00005C (sid:00000000) permanent & global
KDB(1)> e //exit debugger
KDB(1)> r //return to the end of function
```

Chapter 17. Breakpoint and steps subcommands
Chapter 18. Debugger trace points subcommands

Note: Debugger trace points subcommands are specific to the KDB kernel debugger. They are not available in the kdb command.

The subcommands in this category are used to trace the running of a specified address and stop KDB kernel debugger based on conditions. These subcommands include the following:

- bt
- test
- ct
- cat
**bt subcommand**

**Purpose**
The `bt` subcommand traces each a specified address each time it is run.

**Note:** This subcommand is only available within the KDB kernel debugger. It is not included in the kdb command.

**Syntax**
```
btt [\-p | \-v] [address [script]]
```

**Parameters**
- `\-p` Indicates that the trace address is a physical or real address.
- `\-v` Indicates that the trace address is an effective or virtual address.
- `address` Specifies the address of the trace point. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify an address.
- `script` Lists subcommands to be run each time the indicated trace point is run. The script is delimited by quote (") characters and commands within the script are delimited by semicolons (;).

Each time a trace point is encountered, a message is displayed indicating that the trace point was encountered. The displayed message indicates the first entry from the stack. However, this can be changed by using the `script` parameter.

If the `bt` subcommand is invoked with no parameters, the current list of break and trace points is displayed. The number of combined active trace and break points is limited to 32.

It is possible to specify whether the trace address is a physical or a virtual address with the `\-p` and `\-v` options respectively. By default, the KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address.

The `segment id` (sid) is always used to identify a trace point because effective or virtual addresses can have multiple translations in several virtual spaces. When debugging is resumed after a trace point is encountered, **kdb** must reinstall the correct instruction. During this time (one step if no interrupt is encountered), it is possible to miss the trace on other processors.

The script parameter allows a set of **kdb** subcommands to run when a trace point is reached. The set of subcommands comprising the script must be delimited by double quote characters ("). Individual subcommands within the script must be ended by a semicolon (;). One of the most useful subcommands that can be used in a script is the "test subcommand" on page 130. If this subcommand is included in the script, each time the trace point is reached the condition of the test subcommand is checked by the KDB kernel debugger. If the condition is true, a break occurs.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `bt` subcommand:
```
KDB(0)> bt open //enable trace on open()
KDB(0)> bt //display current active traces
0: .open+000000 (sid:00000000) trace {hit: 0}
KDB(0)> e //exit debugger
```
Open routine is traced with a script to display iar and lr registers and to show what is pointed to by the first parameter (r3).

This example shows how to trace and stop when a condition is true. For example, when global data is greater than the specified value, and 923 hits were necessary to reach this condition.
test subcommand

Purpose
The test subcommand can be used in conjunction with the "bt subcommand" on page 128 to break at a specified address when a condition becomes true.

Syntax
test cond

Parameters
cond Specifies the conditional expression that evaluates to a value of either true or false.

The conditional test requires two operands and a single operator. Operands include symbols, hexadecimal values, and hexadecimal expressions. Comparison operators that are supported include: ==, !=, >=, <=, >, and <. Additionally, the bitwise operators ^ (exclusive OR), & (AND), and | (OR) are supported. When bitwise operators are used, any non-zero result is considered to be true.

The syntax for the test subcommand requires that the operands and operator be delimited by spaces. This is very important to remember if the left square bracket ( [ ) alias is used. For example, the subcommand test kernel_heap != 0 can be written as [ kernel_heap != 0 . However, this subcommand is not valid if kernel_heap, !=, and 0 were not preceded by and followed by spaces.

Aliases
[

Example
The following is an example of how to use the [ alias for the test subcommand:
KDB(0)> bt open "[ @sysinfo >= 3d ]" //stop on open() if condition true
KDB(0)> e //exit debugger
... Enter kdb [ @sysinfo >= 3d ]
KDB(1)> bt //display current active trace break points
0: .open+000000 (sid:00000000) trace {hit: 1} {script: [ @sysinfo >= 3d ]}
KDB(1)> dw sysinfo 1 //print sysinfo value
sysinfo+000000: 00000004A

KDB Kernel debugger and kdb command
**cat and ct subcommands**

**Purpose**
The `cat` subcommand erases all trace points. The `ct` subcommand erases individual trace points.

**Syntax**

```
cat
ct slot [-p | -v] Address
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>slot</code></td>
<td>Identifies the slot number for a trace point. This parameter must be a decimal value.</td>
</tr>
<tr>
<td><code>-p</code></td>
<td>Indicates the trace address is a physical or real address.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Indicates the trace address is an effective or virtual address.</td>
</tr>
<tr>
<td><code>Address</code></td>
<td>Identifies the address of the trace point. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify an address.</td>
</tr>
</tbody>
</table>

You can specify the trace point cleared by the `ct` subcommand by a slot number or by an address. By default, KDB kernel debugger chooses the current state of the machine. If the subcommand is entered before VMM initialization, the address is the physical or real address. If the subcommand is entered after VMM initialization, the address is the effective or virtual address.

**Note:** Slot numbers are not fixed. To clear slot 1 and slot 2 type `ct 2; ct 1` or type `ct 1; ct 1`. Do not type `ct 1; ct 2`.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `cat` and the `ct` subcommands:

```
KDB(0)> bt open //enable trace on open()
KDB(0)> bt close //enable trace on close()
KDB(0)> bt readlink //enable trace on readlink()
KDB(0)> bt //display current active traces
  0: .open+000000 (sid:00000000) trace {hit: 0}
  1: .close+000000 (sid:00000000) trace {hit: 0}
  2: .readlink+000000 (sid:00000000) trace {hit: 0}
KDB(0)> ct 1 //clear trace slot 1
KDB(0)> bt //display current active traces
  0: .open+000000 (sid:00000000) trace {hit: 0}
  1: .readlink+000000 (sid:00000000) trace {hit: 0}
KDB(0)> cat //clear all active traces
KDB(0)> bt //display current active traces
No breakpoints are set.
KDB(0)>
```
Chapter 19. Watch DABR subcommands

The subcommands in this category are used to enter the debugger on a load or store instruction. These subcommands include the following:

- `wr`
- `ww`
- `wrw`
- `cw`
- `lwr`
- `lww`
- `lwrw`
- `lcw`
wr, ww, wrw, cw, lwr, lww, lwrw, and lcw subcommands

Purpose

The wr subcommand stops on a load instruction. The ww subcommand stops on a store instruction. The wrw subcommand stops either on a load or a store instruction.

The cw subcommand clears the last watch subcommand. The lwr, lww, lwrw, and lcw subcommands allow you to establish a watchpoint for a specific processor.

Note: These subcommands are only available within the KDB kernel debugger. They are not included in the kdb command.

Syntax

wr [[-e | -p | -v] address [size]]

ww [[-e | -p | -v] address [size]]

wrw[[-e | -p | -v] address [size]]

cw

lwr[[-e | -p | -v] address [size]]

lww [[-e | -p | -v] address [size]]

lwrw [[-e | -p | -v] address [size]]

lcw

Parameters

-e Indicates that the address parameter is an effective or virtual address.
-p Indicates that the address parameter is a physical or real address.
-v Indicates that the address parameter is a virtual or effective address.

address Specifies the address to watch. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address. If the address type is not specified, it is assumed to be an effective address.

size Indicates the number of bytes to watch. This parameter is a decimal value.

A watch register can be used on the Data Address Breakpoint Register (DABR) or HID5 on PowerPC 601 RISC Microprocessor to enter KDB kernel debugger when a specified effective address is accessed. The register holds a double-word effective address and bits to specify load and store operations.

With no parameter, the wr, ww and wrw subcommands print the current active watch subcommand.

The wr, ww, wrw and cw subcommands are global to all processors. The lwr, lww, lwrw and lcw subcommands are local. If no size is specified, the default size is 8 bytes and the address is double-word aligned. If a size is specified, KDB kernel debugger checks the faulting address with the specified range. If no match is found, KDB kernel debugger continues to run.
Aliases
wr → stop-r
ww → stop-w
wrw → stop-rw
cw → stop-cl
lwr → lstop-r
lww → lstop-w
lwrw → lstop-rw
lcw → lstop-cl

Example
The following is an example of how to use the ww, the wr and the cw subcommands:

```
KDB(0)> ww -p emulate_count //set a data break point (physical address, write mode)
KDB(0)> ww //print current data break points
CPU 0: emulate_count+000000 paddr=00238360 size=8 hit=0 mode=W
CPU 1: emulate_count+000000 paddr=00238360 size=8 hit=0 mode=W
KDB(0)> e //exit the debugger
...
Watch trap: 00238360 <emulate_count+000000>
 power_asm_emulate+00013C  stw  r28,0(r30)  r28=0000003A,0(r30)=emulate_count
KDB(0)> ww //print current data break points
CPU 0: emulate_count+000000 paddr=00238360 size=8 hit=1 mode=W
CPU 1: emulate_count+000000 paddr=00238360 size=8 hit=0 mode=W
KDB(0)> wr sysinfo //set a data break point (read mode)
KDB(0)> wr //print current data break points
CPU 0: sysinfo+000000 eaddr=003BA9D0 vsid=00000000 size=8 hit=0 mode=R
CPU 1: sysinfo+000000 eaddr=003BA9D0 vsid=00000000 size=8 hit=0 mode=R
KDB(0)> e //exit the debugger
...
Watch trap: 003BA9D4 <sysinfo+000004>
   .fetch_and_add+000008  lwax  r3,0,r6  r3=sysinfo+000004,r6=sysinfo+000004
KDB(0)> cw //clear data break points
```
Chapter 20. Branch target subcommands

The subcommands in this category provide access on some POWER-based platform processors for target address comparison and trap functions. These subcommands include the following:

- btac
- cbtac
- lbtac
- lcbtac
btac, cbtac, lbtac, lcbtac subcommands

Purpose
Some POWER-based platform processors support an optional branch target address comparison and trap feature. When available, this facility allows for a branch target comparison with some user-provided value with a trap to a specific interrupt vector upon a match. The KDB kernel debugger btac, cbtac, lbtac, and lcbtac subcommands provide access to this facility when it is present. The btac subcommand stops when Branch Target Address Compare (BTAC) is true. The cbtac subcommand clears the last btac subcommand. The cbtac subcommand is global to all processors. Each processor can have different addresses specified or cleared using the local lbtac and lcbtac subcommands.

Note: These subcommands are only available in the KDB kernel debugger. They are not included in the kdb command.

Syntax
btac [ [-e | -p | -v] address]
cbtac
lbtac [ [-e | -p | -v] address]
lcbtac

Parameters
-p Indicates that the address parameter is considered to be a physical or real address.
-v Indicates that the address parameter is considered to be a virtual or effective address.
-e Indicates that the address parameter is considered to be an effective or virtual address.
address Specifies the address of the branch target. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The flags are mutually exclusive. The default flag is -e.

Aliases
No aliases.

Example
The following is an example of how to use the btac, the lbtac and the cbtac subcommands:

KDB(7)> btac open  //set BRAT on open function
KDB(7)> btac  //display current BRAT status
CPU 0: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 1: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 2: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 3: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 4: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 5: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 6: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 7: .open+000000 eaddr=00185354 vsid=00000000 hit=0
KDB(7)> e  //exit the debugger
...
Branch trap: 00185354 <.open+000000>
.sys.call+000000 .bcctrl <.open>
KDB(5)> btac  //display current BRAT status
CPU 0: .open+000000 eaddr=00185354 vsid=00000000 hit=0
CPU 1: .open+000000 eaddr=00185354 vsid=00000000 hit=0

138 KDB Kernel debugger and kdb command
CPU 2: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 3: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 4: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 5: .close+000000 eaddr=00197D40 vsid=00000000 hit=1
CPU 6: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 7: .open+000000 eaddr=001B5354 vsid=00000000 hit=0

KDB(5)> lbtac close //set local BRAT on close function
KDB(5)> e //exit the debugger

Branch trap: 001B5354 <.open+000000>
.sys_call+000000 bctrl <.open>

KDB(7)> e //exit the debugger
...
Branch trap: 00197D40 <.close+000000>
.sys_call+000000 bctrl <.close>

KDB(5)> e //exit the debugger ...
Branch trap: 001B5354 <.open+000000>
.sys_call+000000 bctrl <.open>
KDB(6)> btac //display current BRAT status
CPU 0: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 1: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 2: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 3: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 4: .open+000000 eaddr=001B5354 vsid=00000000 hit=0
CPU 5: .close+000000 eaddr=00197D40 vsid=00000000 hit=1
CPU 6: .open+000000 eaddr=001B5354 vsid=00000000 hit=1
CPU 7: .open+000000 eaddr=001B5354 vsid=00000000 hit=1

KDB(6)> cbtac //reset all BRAT registers
Chapter 21. Namelist and symbols subcommands

The subcommands in this category are used to change namelists and symbols. These subcommands include the following:

- `nm`
- `ts`
- `ns`
- `which`
nm and ts subcommands

Purpose
The nm subcommand translates symbols to addresses. The ts subcommand translates addresses to symbolic representations.

Syntax
nm symbol

ts effectiveaddress

Parameters
symbol Specifies the symbol name.
effectiveaddress Specifies the effective address to be translated. This parameter can be a hexadecimal value or an expression.

Aliases
No aliases.

Example
The following is an example of how to use the nm and the ts subcommands:

KDB(0)> nm __ublock  //print symbol value
Symbol Address : 2FF38400
KDB(0)> ts E3000000  //print symbol name
proc+000000
ns subcommand

**Purpose**
The `ns` subcommand toggles symbolic name translation on and off.

**Syntax**
```
ns
```

**Parameters**
No parameters.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `ns` subcommand:

```
KDB(0)> dc d000 5 //display code at address D000
  ___memcmp+000000 cmpw cr1,r3,r4
  ___memcmp+000004 srwi. r12,r5,2
  ___memcmp+000008 clrlwi r11,r5,1E
  ___memcmp+000010 li r7,0
KDB(0)> ns //disable symbol printing
Symbolic name translation off
KDB(0)> dc d000 5 //display code at address D000
  0000D000 cmpw cr1,r3,r4
  0000D004 srwi. r12,r5,2
  0000D008 clrlwi r11,r5,1E
  0000D010 li r7,0
KDB(0)> ns //enable symbol printing
Symbolic name translation on
KDB(0)>
```
which subcommand

Purpose
The which subcommand displays the name of the kernel source file that contains the address.

Note: The which subcommand is only available in the kdb command.

Syntax
`which | address`

Parameters
`address` Locates the kernel source file that contains the symbol at the specified address and displays the following:
- The symbol corresponding to the address
- The start address of the symbol
- The kernel source file name containing the symbol

Aliases
`wf`

Example
The following is an example of how to use the which subcommand:

```
> which main
  Addr: 0022A700  Symbol: .main
  Name: ../../../../../src/bos/kernel/si/main.c
```
Chapter 22. PCI configuration space and I/O debugging subcommands

The subcommands in this category are used to debug I/O errors and PCI configuration space errors. These subcommands include the following:

- dpcib
- dpcih
- dpciw
- mpcib
- mpcih
- mpciw
- buserr
- businfo
dpcib, dpcih, and dpciw subcommand

Purpose
The dpcib (display PCI byte), dpcih (display PCI halfword), and dpciw (display PCI word) subcommands read data from the PCI Configuration Space.

Syntax
\[
dpcib \, Bid \, PCIslot \, RegOffset
\]
\[
dpcih \, Bid \, PCIslot \, RegOffset
\]
\[
dpciw \, Bid \, PCIslot \, RegOffset
\]

Parameters
- **Bid**: Identifies the Bus Identifier of the PCI bus.
- **PCIslot**: Combines the device number on the PCI bus and the function number on that PCI slot. The combination uses the following formula:
  \[\text{PCIslot} = (\text{device_num} \times 8) + \text{function}\]
- **RegOffset**: Identifies a zero-based byte offset of the register to read in a PCI Configuration Space.

Aliases
No aliases.

Example
The following is an example of how to use the dpcib, the dpcih, and dpciw subcommands:

```
KDB(0)> businfo //get PCI bus id
********** PCI BUSES **********
 ADDRESS    BID    BUS_NUM PHB_UNIT_ID REGIONS
30043400    900000c0 00000000 00000000FE000000 00000004
30043500    900000c1 00000040 00000000FEE00000 00000002
********** OTHER BUSES **********
 ADDRESS    BID    BUS_NUM PHB_UNIT_ID REGIONS
00459AE0    90000040 00000000 0000000000000000 00000001
00459F60    90000100 00000000 0000000000000000 00000002
0045A860    90000300 00000000 0000000000000000 00000001
KDB(0)> dpcib 900000c0 01 4 //display byte of data
00000104:    46
KDB(0)> dpcih 900000c0 01 4 //display halfword of data
00000104:    4600
KDB(0)> dpciw 900000c0 01 4 //display word of data
00000104:    46008022
```
mpcib, mpci, and mpci subcommands

Purpose
The mpcib (modify PCI byte), mpci (modify PCI halfword), and mpci (modify PCI word) subcommands write data to the PCI Configuration Space.

Syntax
mpcib Bid PCIslot RegOffset

mpcii Bid PCIslot RegOffset

mpcii Bid PCIslot RegOffset

Parameters
Bid Identifies the Bus Identifier of the PCI bus.
PCIslot Combines the device number on the PCI bus and the function number on that PCI slot. The combinations uses the following formula:
PCIslot = (device_num * 8) + function
RegOffset Identifies a zero-based byte offset of the register to read in a PCI Configuration Space.

These commands are interactive and each modification is entered one-by-one. The first unexpected input stops modification. A period (.), for example, can be used to indicate the end of the data.

Aliases
No aliases.

Example
The following is an example of how to use the mpcib, the mpci, and the mpci subcommands:
KDB(0)> businfo //get PCI bus id

*********** PCI BUSSES  ***********
ADDRESS BID BUS_NUM PHB_UNIT_ID REGIONS
30043400 900000C0 00000000 00000000F0000000 00000004
30043500 900000C1 00000040 00000000FEE00000 00000002
*********** OTHER BUSSES ***********
ADDRESS BID BUS_NUM PHB_UNIT_ID REGIONS
00459AE0 90000040 00000000 0000000000000000 00000001
00459F60 90000010 00000000 0000000000000000 00000002
0045AB60 90000030 00000000 0000000000000000 00000001
KDB(0)> dpci 900000c0 80 10 //display word of data
00000810: 01FF0F00
KDB(0)> mpci 900000c0 80 10 //modify word
00000810: 01FF0F00 = ffffffff
00000814: 00A010C0 = .
KDB(0)> dpci 900000c0 80 10 //display new word of data
00000810: E1FFFFFF
KDB(0)> mpci 900000c0 80 10 (reset word)
00000810: E1FFFFF = 01FF0F00
00000814: 00A010C0 = .
KDB(0)> dpci 900000c0 80 10 //display reset word
00000810: 01FF0F00
KDB(0)> mpci 900000c0 80 10 //modify specifying bytes
00000810: 01 = ff
00000811: F0 = ff
00000812: FF = ff
00000813: 00 = ff
00000814: 00 = .
KDB(0)> dpciw 900000c0 80 10 //display new word of data
00008010: E1FFFFFF
KDB(0)> mpciw 900000c0 80 10 //reset word
00008010: E1FFFFFF = 01F0FF00
00008014: 00A010C0 = .
KDB(0)> dpciw 900000c0 80 10 //display reset word
00008010: 01F0FF00
KDB(0)> mpcih 900000c0 80 10 //modify specifying halfwords
00008010: 01F0 = ffff
00008012: FF00 = ffff
00008014: 00A0 = .
KDB(0)> dpciw 900000c0 80 10 //display new word of data
00008010: E1FFFFFF
KDB(0)> mpciw 900000c0 80 10 //reset word
00008010: E1FFFFFF = 01F0FF00
00008014: 00A010C0 = .
KDB(0)> dpciw 900000c0 80 10 //display reset word
00008010: 01F0FF00
KDB(0)>
buserr subcommand

Purpose
The buserr subcommand allows PCI bus error injection and manual exercise of EEH capabilities on a PCI slot.

Syntax
buserr bid slot [operation] [function] [bus_addr]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bid</td>
<td>Specifies the bus id. It must be a hexadecimal value.</td>
</tr>
<tr>
<td>slot</td>
<td>Specifies the slot number. It must be a hexadecimal value.</td>
</tr>
<tr>
<td>operation</td>
<td>Specifies the operation code. Accepted values are:</td>
</tr>
<tr>
<td></td>
<td>• 1 - Query slot capabilities and slot state. Displays the state of a slot and information about whether EEH is supported by the slot.</td>
</tr>
<tr>
<td></td>
<td>• 2 - Set slot state. Allows enabling or disabling EEH on a slot or enabling load, store or DMA operation.</td>
</tr>
<tr>
<td></td>
<td>• 3 - Inject a bus error. Performs error injection on a specified bus and slot at a given bus address. The errors can be injected in either memory, I/O or configuration address spaces of a PCI bus. Also, the errors can be on a load or store operation.</td>
</tr>
<tr>
<td></td>
<td>• 4 - Reset slot. This is a way to recover from an EEH event. This operation code can be used to assert and deassert the reset signal on the bus. The reset signal should be asserted for at least 100 milliseconds before deasserting it.</td>
</tr>
<tr>
<td></td>
<td>• 5 - Configure PCI bridge on the adapter. Allows the bridge on an adapter to be configured following a slot reset. This is a required step in complete error recovery for the bridged-adapters such as Ethernet cards.</td>
</tr>
</tbody>
</table>
function
Specifies the function code. It must be a hexadecimal value. Function codes are dependent on the operation code. The available function codes are:

- Operation code 1 - Query slot capabilities and slot state. There are no function codes available.
- Operation code 2 - Set slot state:
  - 0 - Disable EEH
  - 1 - Enable EEH
  - 2 - Enable load/store
  - 3 - Enable DMA
- Operation code 3 - Inject a bus error:
  - 0 - Load to PCI Memory Address Space - inject an Address Parity Error
  - 1 - Load to PCI Memory Address Space - inject a Data Parity Error
  - 2 - Load to PCI I/O Address Space - inject an Address Parity Error
  - 3 - Load to PCI I/O Address Space - inject a Data Parity Error
  - 4 - Load to PCI Configuration Space - inject an Address Parity Error
  - 5 - Load to PCI Configuration Space - inject a Data Parity Error
  - 6 - Store to PCI Memory Address Space - inject an Address Parity Error
  - 7 - Store to PCI Memory Address Space - inject a Data Parity Error
  - 8 - Store to PCI I/O Address Space - inject an Address Parity Error
  - 9 - Store to PCI I/O Address Space - inject a Data Parity Error
  - A - Store to PCI Configuration Space - inject an Address Parity Error
  - B - Store to PCI Configuration Space - inject a Data Parity Error
  - C - DMA read to PCI Memory Address Space - inject an Address Parity Error
  - D - DMA read to PCI Memory Address Space - inject a Data Parity Error
  - E - DMA read to PCI Memory Address Space - inject a Master Abort Error
  - F - DMA read to PCI Memory Address Space - inject a Target Abort Error
  - 10 - DMA write to PCI Memory Address Space - inject an Address Parity Error
  - 11 - DMA write to PCI Memory Address Space - inject a Data Parity Error
  - 12 - DMA write to PCI Memory Address Space - inject a Master Abort Error
  - 13 - DMA write to PCI Memory Address Space - inject a Target Abort Error
- For operation code 4 - Reset slot:
  - 0 - Deactivate Reset
  - 1 - Activate Reset
- For operation code 5 - Configure PCI bridge on the adapter. There are no function codes available.

bus_addr
Specifies the bus address. bus_addr is only used with operation code 3 - Inject a bus error. bus_addr must be a hexadecimal value.

Aliases
No aliases.

Example
The following is an example of how to use the buserr subcommand:

```
KDB(0) > buserr 900000d5 8 1  //query state of slot and if EEH supported
```

Query Slot Capabilities And Slot State
--------------------------------------
Reset State: Reset deactive, EEH not stopped
Slot Capabilities: EEH supported
Success
Select an Operation Code
1) Query Slot Capabilities And Slot State
2) Set Slot State
3) Inject a bus error
4) Reset Slot
5) Configure PCI Bridge on the Adapter
99) Exit

Enter your choice: 99
KDB(0)> buserr 9000000d5 8 4 1 //assert reset

Reset Slot
-------------
Success

Select an Operation Code
1) Query Slot Capabilities And Slot State
2) Set Slot State
3) Inject a bus error
4) Reset Slot
5) Configure PCI Bridge on the Adapter
99) Exit

Enter your choice: 99
KDB(0)> buserr 9000000d5 8 4 0 //deassert reset

Reset Slot
-------------
Success

Select an Operation Code
1) Query Slot Capabilities And Slot State
2) Set Slot State
3) Inject a bus error
4) Reset Slot
5) Configure PCI Bridge on the Adapter
99) Exit

Enter your choice: 99
KDB(0)> buserr 9000000d5 8 3 0 0xf8000000 //inject an address parity error

Inject a bus error
-------------------
Success

Select an Operation Code
1) Query Slot Capabilities And Slot State
2) Set Slot State
3) Inject a bus error
4) Reset Slot
5) Configure PCI Bridge on the Adapter
99) Exit

Enter your choice: 99
businfo subcommand

Purpose
The `businfo` subcommand displays information about all registered buses or about a specified bus.

Syntax
`businfo [-a | -b Bid | eaddr]`

Parameters
- `-a` Displays data for all valid buses.
- `-b Bid` Displays data for bus specified by bus id `Bid`.
- `eaddr` Displays data for the bus at this address.

Aliases
No aliases.

Example
The following is an example of how to use the `businfo` subcommand:

```plaintext
KDB(0)> businfo //display summary
************* PCI BUSES *************
ADDRESS  BID  BUS_NUM  PHB_UNIT_ID  REGIONS
30043400 900000C0 00000000 00000000FEF00000 00000004
30043500 900000C1 00000040 00000000FEE00000 00000002
************* OTHER BUSES *************
ADDRESS  BID  BUS_NUM  PHB_UNIT_ID  REGIONS
00459AE0 90000040 00000000 0000000000000000 00000001
00459F60 90000100 00000000 0000000000000000 00000002
0045AB60 90000300 00000000 0000000000000000 00000001
KDB(0)> businfo -b 900000C0 //display details specifying bus id
next = 00000000  bid = 900000C0
d_map_init = 021D4B08  disable_io = 00000000
num_regions = 00000004
ioaddr[0] = 00000000F8000000  ioaddr[1] = 0000000000000000
bus_specific_data = 00000000  PHB_Unit_ID = 00000000FEF00000
bmap = 00000000
eeh_init = 021D4B14  eeh_init_multifunc = 021D4B04
reserved3 = 00000000  reserved4 = 00000000
KDB(0)> businfo 00459AE0 //display details specifying address
next = 00000000  bid = 90000040
d_map_init = 00000000  disable_io = 00000000
num_regions = 00000001
ioaddr[0] = 0000000000000000  ioaddr[1] = 0000000000000000
bus_specific_data = 00000000  PHB_Unit_ID = 00000000FEF00000
```
bmap = 00000000
eeh_init = 00000000    eeh_init_multifunc = 00000000
reserved3 = 00000000  reserved4 = 00000000

KDB(0)> businfo -a     //display details for all valid buses
********** PCI BUSES **********
Printing Hash bucket 00000000
---------------------------------
next = 00000000  bid = 900000C0
d_map_init = 021D4B08  disable_io = 00000000
num_regions = 00000004
ioaddr[0] = 00000000F8000000  ioaddr[1] = 0000000000000000
bus_specific_data = 00000000  PHB_Unit_ID = 00000000FEF00000
bmap = 00000000
eeh_init = 021D4B14    eeh_init_multifunc = 021D4BD4
reserved3 = 00000000  reserved4 = 00000000

Printing Hash bucket 00000001
---------------------------------
next = 00000000  bid = 900000C1
(0)> more (^C to quit) ? ^C         //interrupt
Chapter 23. Display kernel data structures subcommands

The subcommands in this category are used to print the var and drvars structure and the system configuration of a machine and to display information about IPL control blocks, interrupt handler tables and device switch tables. These subcommands include the following:

- var
- drvars
- ipl
- dev
- intr
### var subcommand

#### Purpose
The `var` subcommand prints the `var` structure and the system configuration of the machine.

#### Syntax
```
var
```

#### Parameters
No parameters.

#### Aliases
No aliases.

#### Example
The following is an example of how to use the `var` subcommand:

```
KDB(7)> var //print var information
var_hdr.var_vers..... 00000000  var_hdr.var_gen..... 00000045
var_hdr.var_size..... 00000030  v_iostrun............ 00000001  v_leastpriv............ 00000000
v_autost............ 00000001  v_memscrub............ 00000000
v_maxup.............. 200  v_mbufhw............. 20  v_mbufhw........... 32768
v_bufhw............. 0  v_mbufl............. 0  v_mbufl........... 0
v_clist.............. 16384  v_fullcore........... 00000000
v_ncpus............... 8  v_ncpus_cfg............. 8
v_initvl.............. 0  v_lock.............. 200  ve_lock............. 0003FA18
v_file.............. 2303  ve_file............. 0042EFE8  file+01AFD0
v_proc.............. 131072  ve_proc............. E3050000  proc+050000
vb_proc............. E3000000  proc+000000
v_thread............ 262144  ve_thread........... E6046F80  thread+046F80
vb_thread.......... E6000000  thread+000000

VMM Tunable Variables:

minfree............... 120  maxfree............... 128
minperm............... 12572  maxperm............... 51488
pfrsvdb1ks............ 13076
(7)> more (^C to quit) //continue
npswarn............... 512  npskill............... 128
minpgahead........... 2  maxpgahead........... 8
maxptblks............... 4  numsched............. 4
htabscale............ FFFFFFFF  aptscale........... 00000000
pd_npages............... 00080000

_SYSTEM_CONFIGURATION:

architecture..... 00000002  POWER_PC
implementation..... 00000010  POWER_604
version............. 00040004
width............... 00000020  ncpus............. 00000008
icache_attrib.... 00000001  CACHE separate I and D
icache_size...... 00040000  dcache_size...... 00040000
icache_asc....... 00000004  dcache_asc....... 00000004
icache_block..... 00000020  dcache_block..... 00000020
icache_line..... 00000040  dcache_line..... 00000040
L2_cache_size.... 00100000  L2_cache_asc.... 00000001
tlb_attrib........ 00000001  TLB separate I and D
tlb_size........ 00000040  dtlb_size........ 00000040
```
Chapter 23. Display kernel data structures subcommands

```
itlb_asc........ 00000002 dtlb_asc........ 00000002
priv_lck_cnt.... 00000000 prob_lck_cnt.... 00000000
resv_size....... 00000020 rtc_type........ 00000002
virt_alias....... 00000000 cach_cong........ 00000000
model_arch...... 00000001 model_impl...... 00000002
Xint............ 000000A0 Xfrac............ 00000003
```
drvars subcommand

Purpose
The `drvars` subcommand displays the global state of Dynamic Reconfiguration (DR) from the `drvars` structure, and displays state about any current DR operation from the `drparms` and `drvars` structures.

Syntax
`drvars`

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the `drvars` subcommand:

```
KDB(0)> drvars
DRparms:

drp_operation...... 00000000
drp_op_idx......... FFFFFFFF
drp_phase.......... FFFFFFFF
drp_errno.......... 00000000
drp_secs.......... 00000000
drp_flags.......... 00000000
drp_pid........... FFFFFFFF
drp_trb........... @ F10010F003FFC280 KERN_heap+3FFC280
drp_timeout....... @ 0000000003A63398 drparms+000028

DRvars:

drbits............ 00000000
flags............. 00000000
lmb_addr.......... 0000000270000000
lmb_size.......... 0000000100000000
RMO_size.......... 0000000400000000
sys_lmbsize....... 0000000100000000
max_num_lmb....... 00000024
actual_num_lmb.... 00000024
fixed_nfr......... 0000000000000000
dead_nfrs........ 00000000
lrudr_running..... 00
gencount.......... 0000000000000006
l_cpuX............ 00000000
l_cpuX_halted.... 00000000
l_cpuY............ 00000000
n_mpcs............ 00000000
gserver........... 00000000
server............ 00000000
trace............. 00000000
```
ipl subcommand

Purpose
The ipl subcommand displays information about IPL control blocks.

Syntax
ipl [* | cpu index]

Parameters

* Displays summary information for all CPUs.

cpu Specifies the CPU number for the IPL control block to be displayed. The CPU is specified as a decimal value.

index Displays the specified index.

Aliases
iplcb

Example
The following is an example of how to use the ipl subcommand:

KDB(4)> ipl * //print ipl control blocks
  INDEX  PHYS_ID INT_AREA ARCHITECT IMPLEMENT VERSION

  0038CD0  0 00000000 FF100000 00000002 00000008 00100005
  0038C098 1 00000001 FF100080 00000002 00000008 00100005
  0038EE60 2 00000002 FF100100 00000002 00000008 00100005
  0038EF28 3 00000003 FF100180 00000002 00000008 00100005
  0038EFF0 4 00000004 FF100200 00000002 00000008 00100005
  0038F0B8 5 00000005 FF100280 00000002 00000008 00100005
  0038F248 6 00000006 FF100300 00000002 00000008 00100005
  0038F248 7 00000007 FF100380 00000002 00000008 00100005

KDB(4)> ipl //print current processor information

Processor_INFO 4 [0038EFF0]

num_of_structs........00000008 index..................00000004
struct_size..........0000000C per_buc_info_offset.......00001000
proc_int_area.........FF100200 proc_int_area_size.....00000010
processor_present....00000001 test_run..............0000000A
test_stat............00000000 link..................00000000
link_address.........00000000 phys_id..............00000004
architecture.........00000002 implementation........00000008
version...............00010005 width..............00000002
cache_attrib..........00000003 coherency_size........00000020
resv_size............00000020 icache_block........00000020
dcache_block.........00000020 icache_size........00000800
dcache_size..........00000800 icache_line........00000000
dcache_line..........00000040 icache_asc...........00000008
dcache_asc...........00000008 L2_cache_size........00010000
L2_cache_asc.........00000011 tlb_attrib...........00000003
itlb_size............00000100 dtlb_size...........00000100
itlb_asc.............00000002 dtlb_asc...........00000002
slt_attrb...........00000000 islb_size...........00000000
dlsb_size............00000000 islb_asc............00000000
(4)> more (^c to quit)? //continue
ds1b_asc.............00000000 priv_lck_cnt...........00000000
prob_lck_cnt.........00000000 rtc_type............00000001
rtcXint..............00000000 rtcXfrac...........00000000
busCfreq_HZ...........00000000 tbCfreq_HZ.........00000000
System info [0038E534]
num_of_procs...........00000008 coherency_size........00000020
resv_size...............00000020 arb_cr_addr...........00000000
phys_id_reg_addr........00000000 num_of_bsrr...........00000000
bsrr_addr...............00000000 tod_type..............00000000
todr_addr..............FF0000C0 rsr_addr............FF62006C
pkssr_addr............FF620064 prcr_addr...........FF620060
sssr_addr...........FF001000 sir_addr.............FF100000
scr_addr...............00000000 dscr_addr...........00000000
nvram_size..........000022000 nvram_addr........FF600000
vpd_rom_addr........00000000 ipl_rom_size........00100000
ipl_rom_addr..........07F00000 g_mfrr_addr........FF107F80
g_tb_addr...............00000000 g_tb_type...........00000000
g_tb_mult...............00000000 SP_Error_Log_Table.....0001C000
pcccrr_addr...........00000000 spocr_addr........FF620068
pfeivr_addr...........FF00100C access_id_waddr........00000000
loc_waddr...............00000000 access_id_raddr........00000000
(4)> more (^C to quit) ? //continue
loc_raddr...............00000000 architecture...........00000001
implementation..........00000002 pkg_descriptor........rs6ksmp
KDB(4)>
devsw subcommand

Purpose
The devsw subcommand displays device switch table entries.

Syntax
\texttt{devsw [major | address]}

Parameters
\begin{itemize}
  \item \texttt{major} Indicates the specific device switch table entry to be displayed by the major number. This is a hexadecimal value.
  \item \texttt{address} Specifies the effective address of a driver. The device switch table entry with the driver closest to the indicated address is displayed. The specific driver is indicated. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
\end{itemize}

Aliases
\texttt{dev}

Example
The following is an example of how to use the dev alias for the devsw subcommand:

\begin{verbatim}
KDB(0)> dev
Slot address 054F5040
 MAJ#001 OPEN   CLOSE  READ  WRITE
   .syopen   .nulldev  .syread  .sywrite
   IOCTL     STRATEGY TTY    SELECT
   .syioctl  .nodev    00000000 .sysel ect
   CONFIG    PRINT     DUMP   MPX
   .nodev    .nodev    .nodev  .nodev
   REVOKE    DSOPT R   SELPTR OPTS
   .nodev    00000000  00000000 00000002

Slot address 054F5080
 MAJ#002 OPEN   CLOSE  READ  WRITE
   .nulldev  .nulldev  .mmread  .mmwrite
   IOCTL     STRATEGY TTY    SELECT
   .nodev    .nodev    00000000 .nodev
   CONFIG    PRINT     DUMP   MPX
   .nodev    .nodev    .nodev  .nodev
   REVOKE    DSOPT R   SELPTR OPTS
   .nodev    00000000  00000000 00000002
\end{verbatim}

(0)> more (^C to quit) ? ^C //quit
KDB(0)> devsw 4 //device switch of major 0x4
Slot address 05640100
 MAJ#004 OPEN   CLOSE  READ  WRITE
   .conopen  .conclose  .conread  .conwrite
   IOCTL     STRATEGY TTY    SELECT
   .conioctl .nodev    00000000 .conse lect
   CONFIG    PRINT     DUMP   MPX
   .conconfig .nodev    .nodev  .conmp x
   REVOKE    DSOPT R   SELPTR OPTS
   .conrevoke 00000000  00000000 00000006
\end{verbatim}
intr subcommand

Purpose
The intr subcommand prints a summary for entries in the interrupt handler table if no parameter or a slot number is entered.

Syntax
intr [ slot \ address]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Specifies the slot number in the interrupt handler table. This value must be a decimal value.</td>
</tr>
<tr>
<td>address</td>
<td>Specifies the effective address of an interrupt handler. Symbols, hexadecimal values or hexadecimal expressions can be used to specify the address.</td>
</tr>
</tbody>
</table>

If no parameter is entered, the summary contains information for all entries. If a slot number is specified, only the selected entries are displayed. If an address parameter is entered, detailed information is displayed for the specified interrupt handler.

Aliases
No aliases.

Example
The following is an example of how to use the intr subcommand:

```
KDB(0)> intr //interrupt handler table
SLT INTRADDR HANDLER TYPE LEVEL PRIO BID FLAGS
i_data+000068 1 055DF0A0 00000000 0000 00000003 0000 00000000 0000
i_data+000068 1 00364F88 00090584 0000 00000001 0000 00000000 0000
i_data+000068 1 003685B0 00090584 0001 00000000 0000 00000000 0000
i_data+000068 1 019E7D48 019E7BF0 0000 00000001 0000 00100000 0000
i_data+0000E0 16 055DF060 00000000 0001 00000001 0000 00000000 0000
i_data+0000E0 16 00368718 000A24D8 0000 00000000 0000 00000000 0000
i_data+0000E0 16 019E7D48 019E7BF0 0000 00000001 0000 00100000 0000
i_data+0000F0 18 055DF100 00000000 0001 00000000 0001 00000000 0000
i_data+0000F0 18 05B3BC00 01A55018 0000 00000002 0001 00100000 0000
i_data+000120 24 055DF0C0 00000000 0001 00000000 0004 00000000 0000
i_data+000120 24 003685B0 00090584 0001 00000000 0000 00000000 0000
i_data+000120 24 019E7D48 019E7BF0 0000 00000001 0000 00000000 0000
i_data+000140 28 055DF160 00000000 0001 00000001 0003 00000000 0010
i_data+000140 28 0A145000 01A741AC 0001 0000000C 0003 00000000 0100
i_data+000150 30 055DF0E0 00000000 0001 00000000 0003 00000000 0010
i_data+000150 30 055FC000 019E7A8A 0001 00000000 0003 00000000 0010
i_data+000160 32 055DF080 00000000 0001 00000002 0000 00000000 0000
i_data+000160 32 00368734 000A24D8 0001 00000000 0000 00000000 0000
i_data+0004E0 144 055DF020 00000000 0002 00000000 0000 00000000 0011
i_data+0004E0 144 003685B0 00090380 0002 00000002 0000 00000000 0011
i_data+000530 154 055DF0A0 00000000 0002 FFFFFFFF 000A 00000000 0011
i_data+000530 154 003685B0 00090380 0002 00000002 000A 00000000 0011
KDB(0)> intr //interrupt handler slt 1
SLT INTRADDR HANDLER TYPE LEVEL PRIO BID FLAGS
addr............ 00368560 handler...... 00090380 i_hwassist_int+000000
bid............ 00000000 bus_type...... 00000002 PLANAR
```
next........... 00000000 flags........ 00000011 NOT_SHARED MPSAFE
level........... 00000002 priority...... 00000000 INTMAX
i_count........ 00000014
KDB(0)>
Chapter 24. Display VMM subcommands

The subcommands in this category can be used to display VMM information. These subcommands include the following:

- ames
- apf
- frameset
- free
- freelist
- ipc
- rtipc
- rtipcd
- lka
- lkha
- lkwh
- mempool
- pfd
- pfhdata
- pft
- swhat
- pvt
- pta
- pte
- map
- rvsid
- scb
- segst64
- sr64
- ksp
- ste
- vmbufst
- vmaddr
- vmdmap
- vmint
- vmker
- vmlocks
- vmlog
- vmpool
- vmstat
- vmthrgio
- vmwait
- vsidd
- vsidm
- zproc
• `drlist`
• `drlist`
ames subcommand

Purpose
The ames subcommand provides options for the display of the process address map for either the current process, a specified process, or a specified address map.

Syntax
ames [menu options]

Parameters
menu options Menu options and parameters can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without arguments, menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known they can be entered as subcommand arguments.

Aliases
No aliases.

Example
The following is an example of how to use the ames subcommand:

KDB(0)> ames
VMM AMEs
Select the ame to display by:
1) current process
2) specified process
3) specified address map
Enter your choice: 2
Enter the process id: 0326E
Switch to proc: E2006400

VMM address map, address D0000000
previous entry (vme_prev) : 00000040
next entry (vme_next) : 00000040
start of range (min_offset) : 30000000
end of range (max_offset) : F0000000
number of entries (nentries) : 00000001
size (size) : 00100000
non-directed map. (min_offset2) : 30000000
reference count (ref_count) : 00000001
hint (hint) : 00000004
first free hint (first_free) : 00000004
entries pageable (entries_pageable) : 00000000

VMM map entry, address D00000040
previous entry (vme_prev) : 00000000
next entry (vme_next) : 00000000
start address (vme_start) : 30000000
end address (vme_end) : 30100000
object (vnode ptr) (object) : 14F1B380
page num in object (obj_pno) : 00000000cur protection (protection) : 00000003
max protection (max_protection) : 00000007
inheritance (inheritance) : 00000000
source sid (source_sid) : 0000E347
mapping sid (mapping_sid) : 00003434
Enter the sid (in hex): 00008344

VMM SCB Addr B0489BE C Index 00000344 of 0000050B Segment ID: 00008344

//MAPPING SEGMENT
> (_segtype)..... mapping segment
segment info bits (_sibits) : 10000000
default storage key (_defkey) : 0
starting ame (same) : 00000040
ending ame (eame) : 00000040
hint ame (hame) : 00000040
waitlist for change (msegwait) : 00000000
> (mappings).... mappings exist
sibling mmap fork seg (sibling) : 00000000
class ID (classid) : 00000000 0
physical attachments (_att) : 00000000
mmap reference count (refcnt) : 00000001
non-fblu pageout count (npopages) : 0000
xmem attach count (xmemcnt) : 0000
pages in real memory (npages) : 00000000
pinned pages in memory (npinpages) : 00000000
lru pageout count (npopages) : 00000000
proc pointer (proc) : E2006400
(0)> more (^C to quit) ?
page frame at head (sidlist) : FFFFFFFF
max assigned page number (maxvpn) : FFFFFFFF
lock (lock) : 0B0489C44 00000000
KDB(0)>
apt subcommand

Purpose
The apt subcommand provides options for display of information from the alias page table.

Syntax
apt [menu options]

Parameters
menu options Menu options and parameters can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without arguments, menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known, they can be entered as subcommand arguments.

Aliases
No aliases.

Example
The following is an example of how to use the apt subcommand:

Example:
KDB(0)> apt
VMM APT
Select the APT function:
  1) display by index
  2) display by sid,pno
  3) display by page frame
  4) count valid, free
  5) count free from pf_aptfree
  6) count valid from AHAT
  7) display free list
Enter your choice: 1
Enter the index (in hex): 0
VMM APT Entry 00000000 of 00010000
> valid
segment identifier (sid) : 0002A015
page number (pno) : 0000
page frame (nfr) : 00000000
protection key (key) : 3
storage control attr (wimg) : 2
next on hash (next) : FFFF
next on alias list (anext) : FFFF
next free/pin count (free) : 0001
KDB(0)> apt 2
Enter the sid (in hex): 2a015
Enter the pno (in hex): 0

VMM APT Entry 00000000 of 00010000
> valid
segment identifier (sid) : 0002A015
page number (pno) : 0000
page frame (nfr) : 00000000
protection key (key) : 3
storage control attr (wimg) : 2
next on hash (next) : FFFF
next on alias list (anext): FFFF
next free/pin count (free) : 0001
KDB(0)> apt 4
There are 10000 APT slots allocated.
12 are valid
FFEE are free
KDB(0)> apt 7
KDB Kernel debugger and kdb command
frameset subcommand

Purpose
The frameset displays information about VMM frame sets.

Syntax
frameset [frs_id]

Parameters
frs_id
Can be the * character to specify a summary of the frame set table should be displayed. Or, it can be a specific frameset id to indicate detailed information about the specific frameset should be displayed.

Note: The frameset subcommand requires a parameter.

Aliases
frs

Example
The following is an example of how to use the frameset subcommand:

KDB(1)> frameset *

VMP MEMP FR S NEXT_FRS NB_PAGES NUMFRB
memp_frs+000000 00 00 00 00000001 0013B28C 00128CFB
memp_frs+000080 00 00 001 FFFFFFFF 0013B2BA 00128D11
KDB(1)> frameset 1

Frame Set [1] [00000000000EC7080]

> valid
freefwd (freefwd) : 000000000009C7D5
freebwd (freebwd) : 000000000009C8F3
free nfr lock @ 00000000000EC7080 00000000
free frames (numfrb) : 0000000000128D11
number of frames (nb_frame) : 000000000013B2BA
next frameset (next_frs) : FFFFFFFF
owning mempool (memp_id) : 00000000
owning vmpool (vmpool_id) : 00000000
KDB(1)>
free subcommand

Purpose
The free subcommand counts the number of free page frames.

Syntax
free

Parameters
No parameters are supported for the free subcommand.

The free subcommand counts and displays the number of free page frames, on a vmpool/frameset basis.

Note: The time it takes for this command to complete depends on the amount of system memory being considered. Noticeable delays are not unusual.

Aliases
No aliases.

Example
The following is an example of how to use the free subcommand:

KDB(1)> free
VMPOOL: 00
frame set 0 : 128CFB free frames
frame set 1 : 128011 free frames
KDB(1)>
freelist subcommand

Purpose
The freelist subcommand displays VMM free list information.

Syntax
freelist [frs_id]

Parameters
frs_id Specifies the frameset identifier for which you want to display VMM free list information.

The freelist subcommand requires an frs_id parameter to identify the particular frameset to examine. The list of all page frames on the free list for that frameset is then displayed.

Note: The longer the length of the free list, the more time this subcommand takes to complete.

Aliases
No aliases.

Example
The following is an example of how to use the freelist subcommand:

```
KDB(0)> freelist 1
00000261A5 - 00000261B5 - 00000261A3 - 00000261B1 - 00000261AF - 00000261AD -
00000261A8 - 00000261A9 - 00000261A7 - 0000026198 - 00000261A1 - 000002619F -
000002619D - 0000026189 - 0000026199 - 0000026197 - 0000026195 - 0000026193 -
0000026191 - 000002618F - 000002618D - 000002618B - 0000026183 - 0000026187 -
0000026185 - 0000024951 - 0000024AFD - 0000024AEB - 0000024D09 - 000002616D -
0000026121 - 0000024B9B - 0000024B9D - 000002613D - 0000024D11 - 0000024D15 -
0000024AFB - 000002617D - 0000024BC3 - 0000026178 - 0000024D77 - 0000026179 -
<snip>
00000261FD - 00000261FB - 00000261F9 - 00000261F7 - 00000261F5 - 00000261F3 -
00000261F1 - 00000261EF - 00000261ED - 00000261EB - 00000261E9 - 00000261E7 -
00000261E5 - 00000261E3 - 00000261E1 - 00000261DF - 00000261DD - 00000261DB -
00000261D9 - 00000261D7 - 00000261D5 - 00000261D3 - 00000261D1 - 00000261CF -
00000261CD - 00000261CB - 00000261C9 - 00000261C7 - 00000261C5 - 00000261C3 -
00000261C1 - 00000261BF - 00000261BD - 00000261BB - 00000261B9 - 00000261B7 -
FBANCH
2905E free frames
KDB(0)>
```
ipc subcommand

Purpose
The ipc subcommand reports interprocess communication facility information.

Syntax
ipc [menu options]

Parameters
menu options Menu options and parameters can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known, you can enter them as subcommand parameters.

ipc 1 [1..3] to print message queue information
ipc 2 [1..2] to print shared memory information
ipc 3 [1..2] to print semaphore information

Aliases
No aliases.

Example
The following is an example of how to use the ipc subcommand:

KDB(0)> ipc
IPC info
Select the display:
  1) Message Queues
  2) Shared Memory
  3) Semaphores
Enter your choice: 1
  1) all msqid_ds
  2) select one msqid_ds
  3) struct msg
Enter your choice: 1
Message Queue ID 00000000 @ 00000000
uid............ 48454150 gid............ 00043000
 cuiid.......... 00000000 cgid.......... 00000001
mode............ 0000FFBD seq............ 0000
key............. 40000000
msg_first...... 00000000
msg_last....... 00000000
msg_cbytes..... 00000000 msg_qnum...... 00000000
msg_qbytes..... 00000000
msg_lspid...... 00000000
msg_lrpid...... 00000000
msg_stime...... 00000000
msg_rtime...... 00000000
msg_ctime...... 00000000
msg_rwait..... 00000000 msg_wwait..... 00000000
msg_reqevents. 0000
msg_next....... 00000000
msg_prev....... 00000000
orig_msqid.... 00000000 cur_msqid.... 00000000 crid............ 00000000
vhat_next..... 00000000
vhat_prev..... 00000000
rt_ipcx........ 00000000
maxmsg........ 00000000
notify........ NULL
KDB(0)>

Chapter 24. Display VMM subcommands  175
rtipc subcommand

Purpose
The rtipc subcommand reports posix realtime interprocess communication facility information.

Syntax
rtipc [menu options]

Parameters
menu options Identifies menu options and parameters that can be entered along with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data to be displayed.

(0)> rtipc
RTIPC info
Select the display:
1) Message Queues
2) Shared Memory
3) Semaphores
4) Message Queue Name Table
5) Shared Memory Name Table
6) Semaphore Name Table
Enter your choice:

Reported information is related to posix realtime message queues, shared memory and semaphores, and their associated name table.

Note: If the menu selections and required values are known, they can be entered as subcommand parameters.

For realtime ipc objects, displayed data can be selected by object address, index in object table, or realtime ipc name. If selection is by name, the subcommand must be invoked with all its parameters.

(0)> rtipc 1
1) all entries
2) select one entry by address
3) select one entry by index
4) select one entry by name
   (name up to 16 chars, type command in once)
Enter your choice:

For a realtime ipc name table, displayed data can be selected by index in the name table.

(0)> rtipc 4
1) all entries
2) select one entry by index
Enter your choice:

Aliases
No aliases.

Example
The following is an example of how to use the rtipc subcommand:
(0)> rtipc
RTIPC info
Select the display:
  1) Message Queues
  2) Shared Memory
  3) Semaphores
  4) Message Queue Name Table
  5) Shared Memory Name Table
  6) Semaphore Name Table
Enter your choice: 1
  1) all entries
  2) select one entry by address
  3) select one entry by index
  4) select one entry by name
      (name up to 16 chars, type command in once)
Enter your choice: 2
Enter the address (in hex): F10000B08013BD98

RT Message Queue idx 00007E57 @ F10000B08013BD98
next........... 0000000000000000
name........... mymq
sysVid......... 000C7E59
flags.......... 00000001 INUSE
refcnt......... 00000000
msgsize........ 00000400

(0)> rtipc 4 1
00000030 : F10000B080360998
00000061 : F10000B08026A520
00000062 : F10000B08029C458 F10000B08025B520
00000064 : F10000B080279368 F10000B08026F430
0000006A : F10000B080269F80
rtipcd subcommand

Purpose

The rtipcd subcommand reports posix realtime ipc descriptor information.

Syntax

rtipcd [menu options]

Parameters

menu options   Use menu options and parameters with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine which data to display.

Reported information is related to process descriptors of posix realtime message queues and semaphores, and process descriptor hash tables.

```
0)> rtipcd
RTIPC Descriptor info
Select the display:
  1) Message Queue Descriptors
  2) Semaphore Descriptors
  3) Message Queue Descriptor Table
  4) Semaphore Descriptor Table
Enter your choice:
```

For realtime ipc descriptors, displayed data can be selected by descriptor address or descriptor user id.

```
0)> rtipcd 1
  1) select one entry by address
  2) select one entry by user id
Enter your choice:
```

For realtime ipc descriptor tables, displayed data can be selected by hash table index.

```
0)> rtipcd 3
  1) all entries
  2) select one entry by index
Enter your choice:
```

Aliases

No aliases.

Example

The following is an example of how to use the rtipcd subcommand:

```
0)> rtipcd
RTIPC Descriptor info
Select the display:
  1) Message Queue Descriptors
  2) Semaphore Descriptors
  3) Message Queue Descriptor Table
  4) Semaphore Descriptor Table
Enter your choice: 1
  1) select one entry by address
  2) select one entry by user id
Enter your choice: 1
```

KDB Kernel debugger and kdb command
Enter the address (in hex): F100009E189B5C00

RT Message Queue Descriptor @ F100009E189B5C00
next........... F100009E189B5F00
rt_ipcx........ 0001AD34
mq oflags..... 00000003 READ WRITE
mq umqid...... 68000000 idx.. 0034 seq.. 00000000

(0)> rtipcd 3 1
0000001C : F100009E189B57E0
00000034 : F100009E189B5C00 F100009E189B5F00
00000037 : F100009E189B5AE0
Ika subcommand

Purpose
The Ika subcommand displays VMM lock anchor data and data for the transaction blocks in the transaction block table. You can display individual entries of the transaction block table by providing a slot number or an effective address.

Syntax
Ika [slot | effectiveaddress]

Parameters

slot Specifies the slot number in the transaction block table to be displayed. This parameter must be a decimal value.

effectiveaddress Specifies the effective address of an entry in the transaction block table. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

Aliases
lockanch, tblk

Example
The following is an example of how to use the Ika subcommand:

KDB(0)> lka
VMM LOCKANCH lwseg +000000
nexttid...... : 000008210 freetid...... : 00000002 maxtid...... : 00000002
lwptr........ : D0000000 freelock...... : 00000000 morelocks.... : D000C000
syncwait..... : 00000000 tblkwait..... : 00000000 freewait..... : 00000000
lw lock...... @ 006F0BE0 00000000
tblk......... @ 000000024 lockhash..... @ D000A024
   @tblk[0] lwseg +000024
  logtid.... 00000000 next...... 00000000 tid........ 00000000 flag...... 00000000
cpn........ 00000000 ceor....... 00000000 cxor....... 00000000 csn....... 00000000
  waitsid...... 0000000000 waitline.. 00000000 locker.... 00000000 lsidx..... 00000000
  gcw.ellist. 00000000 gcw.owner. 00000000 gcw.lock.. 00000000 gcw.boost. 00000000
logage.... 00000000 waitors... 00000000 cqnext.... 00000000
   @tblk[1] lwseg +000074 tblk[1].cqnext lwseg +0000C4
  logtid.... 0000A72A next...... 00000003 tid....... 00000001 flag...... 0000002D
cpn........ 00001AC6 ceor....... 00000530 cxor....... 1D696F24 csn....... 00000003
  waitsid...... 00000000 waitline.. 00000000 locker.... 00000000 lsidx..... 00000000
  gcw.ellist. FFFFFFFF gcw.owner. 00000000 gcw.lock.. 00000000 gcw.boost. 00000000
logage.... 00000000 waitors... 00000000 cqnext.... D00000C4
flag...... QUEUE COMMIT COMMITTED LEADER
   @tblk[2] lwseg +0000C4
(0)> more (`c to quit)?

KDB(0)>
**Ilkh subcommand**

**Purpose**
The *ilkh* subcommand displays the contents of the VMM lock hash list. The entries for a particular hash chain can be viewed by specifying the slot number or effective address of an entry in the VMM lock hash list.

**Syntax**
```
ilkh [slot | eaddr]
```

**Parameters**
- `slot` Specifies the slot number in the VMM lock hash list. This parameter must be a decimal value.
- `eaddr` Specifies the effective address of a VMM lock hash list entry. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

**Aliases**
`lockhash`

**Example**
The following is an example of how to use the *ilkh* subcommand:
```
KDB(0)> 1kh
  BUCKET  HEAD  COUNT
lkwseg +00F090  22  0000001  1
KDB(0)> 1kh @r3
HASH ENTRY( 1): F100009C0000F03C
KDB(0)> dr r3
r3  : 0000000000000001 00000001
KDB(0)> 1kh 1
HASH ENTRY( 1): F100009C0000F03C
```
Ikw subcommand

Purpose
The Ikw subcommand displays VMM lock words.

Syntax
Ikw [slot | effectiveaddress]

Parameters
- slot: Specifies the slot number of an entry in the VMM lock word table. This parameter must be a decimal value.
- effectiveaddress: Specifies the effective address of an entry in the VMM lock word table. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

If no parameter is entered, a summary of the entries in the VMM lock word table is displayed, one line per entry. If a parameter identifying a particular entry is entered, details are shown for that entry and the following entries on the transaction ID chain.

Aliases
lockword

Example
The following is an example of how to use the Ikw subcommand:

KDB(0)> Ikw

```
  NEXT  TIDNXT  SID  PAGE  TID  FLAGS
  0 lkwseg +00B000 0  0 00000000 0000 0000
  1 lkwseg +00B028 4  3 00030008 0002 0002 WRITE FREE
  2 lkwseg +00B050 3  6 0002B074 0001 0002 WRITE FREE
  3 lkwseg +00B078 1  2 0002B074 0000 0002 WRITE FREE
  4 lkwseg +00B0A0 5  1 00030008 0008 0001 WRITE FREE
  5 lkwseg +00B0C8 7  4 00030008 0003 0001 WRITE FREE
  6 lkwseg +00B0F0 2  0 000100A8 018F 0002 WRITE FREE
  7 lkwseg +00B118 8  0 00000000 0000 0000
  8 lkwseg +00B140 9  0 00000000 0000 0000
  9 lkwseg +00B168 10 0 00000000 0000 0000
 10 lkwseg +00B190 11 0 00000000 0000 0000
 11 lkwseg +00B1B8 12 0 00000000 0000 0000
 12 lkwseg +00B1E0 13 0 00000000 0000 0000
 13 lkwseg +00B208 14 0 00000000 0000 0000
 14 lkwseg +00B230 15 0 00000000 0000 0000
 15 lkwseg +00B258 16 0 00000000 0000 0000
 16 lkwseg +00B280 17 0 00000000 0000 0000
 17 lkwseg +00B2A8 18 0 00000000 0000 0000
 18 lkwseg +00B2D0 19 0 00000000 0000 0000
 19 lkwseg +00B2F8 20 0 00000000 0000 0000
 20 lkwseg +00B320 21 0 00000000 0000 0000
 21 lkwseg +00B348 22 0 00000000 0000 0000
 22 lkwseg +00B370 23 0 00000000 0000 0000
 23 lkwseg +00B398 24 0 00000000 0000 0000
 24 lkwseg +00B3C0 25 0 00000000 0000 0000
 25 lkwseg +00B3E8 26 0 00000000 0000 0000
 26 lkwseg +00B410 27 0 00000000 0000 0000
 27 lkwseg +00B438 28 0 00000000 0000 0000
 28 lkwseg +00B460 29 0 00000000 0000 0000
<snip>
```
KDB(0)> Ikw 3
```
<table>
<thead>
<tr>
<th>NEXT</th>
<th>TIDNXT</th>
<th>SID</th>
<th>PAGE</th>
<th>TID</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>lwseg</td>
<td>+00B078</td>
<td>1</td>
<td>2</td>
<td>00028074</td>
</tr>
<tr>
<td></td>
<td>bits...</td>
<td>20000000</td>
<td>log...</td>
<td>01841588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>home...</td>
<td>00000020</td>
<td>extmem..</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT TIDNXT</td>
<td>SID PAGE</td>
<td>TID</td>
<td>FLAGS</td>
</tr>
<tr>
<td>2</td>
<td>lwseg</td>
<td>+00B050</td>
<td>3</td>
<td>6</td>
<td>00028074</td>
</tr>
<tr>
<td></td>
<td>bits...</td>
<td>10000000</td>
<td>log...</td>
<td>01841588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>home...</td>
<td>00000021</td>
<td>extmem..</td>
<td>00000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEXT TIDNXT</td>
<td>SID PAGE</td>
<td>TID</td>
<td>FLAGS</td>
</tr>
<tr>
<td>6</td>
<td>lwseg</td>
<td>+00B0F0</td>
<td>2</td>
<td>0</td>
<td>000100A8</td>
</tr>
<tr>
<td></td>
<td>bits...</td>
<td>00020000</td>
<td>log...</td>
<td>01851C88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>home...</td>
<td>0000300F</td>
<td>extmem..</td>
<td>00000000</td>
<td></td>
</tr>
</tbody>
</table>
```
mempool subcommand

Purpose
The mempool subcommand displays information about VMM memory pools.

Syntax
mempool [memp_id]

Parameters
memp_id  Is the asterisk (*) character or a memory pool identifier. The asterisk (*) displays a summary of the memory pool table. A specific memory pool identifier displays detailed information about the specific memory pool.

Note: The mempool subcommand requires a parameter.

Aliases
memp

Example
The following is an example of how to use the mempool subcommand:

KDB(1) > mempool *

<table>
<thead>
<tr>
<th>VMP</th>
<th>MEMP</th>
<th>NB_PAGES</th>
<th>FRAMESETS</th>
<th>NUMFRB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>memp frs+040000</td>
<td>00</td>
<td>00276576</td>
<td>000 001</td>
</tr>
</tbody>
</table>

KDB(1) > mempool 0

Memory Pool [0] [00F07000]
Frame Sets:
[00000000] [00EC7000]
[00000001] [00EC7080]

> valid
number of frames (nb_frame) : 0000000000276576
first frame set (first_frs) : 00000000
next memory pool (next) : FFFFFFFF
owning vmpool (vmpool_id) : 00000000

LRU statistics and thresholds
min perm frames (minperm) : 000000000007AA86
max perm frames (maxperm) : 00000000001EAA18
max client frames (maxclient) : 00000000001EAA18
fbiru page-outs (numpout) : 0000000000000000
fbiru remote pg-outs (numremote) : 0000000000000000
num client frames (numclient) : 0000000000000000
compressed segs (numcompress) : 0000000000000000
num perm frames (numperm) : 0000000000001940

(1)> more (^C to quit) ?
comp repage cnt (rpgcnt[RPCOMP]) : 0000000000000000
file repage cnt (rpgcnt[RFPFILE]) : 0000000000000000
freewake (freewake) : 00000000
free frame wait (freewait) : 0000000000000000
v_sync cursor (syncptr) : 00000000
next lru candidate (lruprtr) : 00000000000000021
frames examined (lrucnt) : 0000000000000000
start of bucket (lrumin) : 00000000000000022
end of bucket (lrumax) : FFFFFFFF
LRU bucket size (lrbucket) : 0000000000020000
lru interval head (lrumem) : F100001420000080
nolru interval head (nolru) : F1000014200000C0

184  KDB Kernel debugger and kdb command
index in int array (lruidx) : F100001420000300
lru index for bucket (saveidx) : F100001420000300
force fileonly off (fileonly_off) : 00000000
lru daemon anchor (lru_daemon) : F1000009E14741400
lru request (lru_requested) : 00000000
DR thread id (dr_tid) : FFFFFFFF
lru pf color (lru_pf_color) : 0
LRU lock @ 0000000000F67000: 00000000
KDB(1)>
## pdt subcommand

### Purpose
The `pdt` subcommand displays entries of the paging device table.

### Syntax
```
pdt [*] [slot]
```

### Parameters
- `*` Displays all entries of the paging device table.
- `slot` Specifies the slot number within the paging device table to be displayed. This value must be a hexadecimal value.

An asterisk (`*`) parameter displays all entries in a summary. To display the details for a specific entry, specify the slot number in the paging device table. If no parameter is specified, you are prompted to enter the PDT index you want to display. Detailed data is then displayed for the entered slot and all higher slot numbers.

### Aliases
No aliases.

### Example
The following is an example of how to use the `pdt` subcommand:

```
KDB(0)> pdt *       // display paging device table

SLT NEXTIO DEVICE IOTAIL DMSRVAL IOCNT <name>
vmmdseg+460000 0000 FFFFFFFF 000A0002 FFFFFFFF 00000000 00000000 paging
vmmdseg+4605B0 0010 FFFFFFFF 0607A2C FFFFFFFF 00000000 00000000 remote
vmmdseg+4605DB 0011 FFFFFFFF 000A0007 FFFFFFFF 0002081 00000000 00000000 filesystem
vmmdseg+460630 0012 FFFFFFFF 000A0003 FFFFFFFF 00000000 00000000 log
vmmdseg+460680 0013 FFFFFFFF 000A0004 FFFFFFFF 0003609B 00000000 00000000 filesystem
vmmdseg+460738 0015 FFFFFFFF 000A0006 FFFFFFFF 0034000 00000000 00000000 filesystem
vmmdseg+460790 0016 FFFFFFFF 0607A1C FFFFFFFF 00000000 00000000 remote
vmmdseg+4607E8 0017 FFFFFFFF 000A0009 FFFFFFFF 000122A 00000000 00000000 filesystem
vmmdseg+460840 0018 FFFFFFFF 000A000B FFFFFFFF 0002020 00000000 00000000 filesystem
vmmdseg+460898 0019 FFFFFFFF 000A0008 FFFFFFFF 000140AA 00000000 00000000 filesystem
vmmdseg+4608F0 001A FFFFFFFF 0222D694 FFFFFFFF 00000000 00000000 00000000 remote

KDB(0)> pdt 13 // display paging device table slot 13

PDT address 00460688 entry 0018 of 03FF, type: FILESYSTEM
next pdt on i/o list (nextio) : FFFFFFFF
dev_t or strategy ptr (device) : 000A0004
last frame w/pend I/O (iotail) : FFFFFFFF
free buf struct list (bufstr) : 30861B
.total buf structs (nbufs) : 008A
available (PAGING) (avail) : 0000
JFS disk agsize (agsize) : 0800
JFS inode agsize (iagsize) : 1000
JFS log SCB index (logsidx) : 0008F
JFS fragments per page(fperpage): 1
JFS compression type (comptype): 0
JFS log2 bigalloc mult(bigexp) : 0
disk map srval (dmsrval) : 0003609B
i/o's not finished (iocnt) : 00000000
device wait list (devwait) : 00000000
buffer wait list (bufwait) : 00000000
```
logical volume lock (lock) :0B04606BB 00000000
buffer list lock (buf_lock) :0B04606BC 00000000
flag bits (devflags) : 80000000
max phys Xlation ent (maxphys) : 00000020
SR val for .indirect (indsrval) : 00030098
SR val for .inodes (inosrval) : 00032099
SR val for .inodemap (imsrval) : 0003409A
KDB(0)
pfhdata subcommand

Purpose
The pfhdata subcommand displays virtual memory control variables.

Syntax
pfhdata

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the pfhdata subcommand:

KDB(0)> pfhdata

VMM Control Variables: 80476000 vmmdseg +476000

1st free sid entry (sidfree) : 000004A5
1st delete pending (sidxmem) : 00000000
highest sid entry (hisid) : 0000050C
frames not pinned (pfavail) : 0005965F
app frames free (ppinavail) : 000511B6
lru bucket size (lrbucket) : 00020000
last pdt on i/o list (ioutil) : FFFFFFFF
num of paging spaces (npgspaces) : 00000001
PDT last alloc from (pdtlast) : 00000000
max pgsp PDT index (pdtmaxxpg) : 00000000
PDT free pool list (pdtfree) : 00000000
PDT high watermark (pdtmax) : 0000001B
PDT index of server (pdtserver) : 00000000
scb serial num (nxtscbnum) : 0000060D
num of comp replaces (nreplaced[RPCOMP]) : 00000000
num of file replaces (nreplaced[RFPFILE]) : 00000000
num of comp repages (nrepaged[RPCOMP]) : 00000000
num of file repages (nrepaged[RFPFILE]) : 00000000
min page-ahead (minpgahead) : 00000000
max page-ahead (maxpgahead) : 00000000
sysbr protect key (kerkey) : 00000000
non-ws page-outs (numpermio) : 00000000
free frame wait (freewait) : 00000000
device i/o wait (devwait) : 00000000
extend XPT wait (extendwait) : 00000000
buf struct wait (bufwait) : 00000000
inh/delete wait (deletewait) : 00000000
SIGDANGER level (npswarn) : 00001000
SIGKILL level (npkill) : 00000400
next warn level (nextwarn) : 00000100
next kill level (nextkill) : 00000400
adj warn level (adjwarn) : 00000008
adj kill level (adjkill) : 00000008
cur pdt alloc (npdtdblks) : 00000002
max pdt alloc (maxpdtdblks) : 00000004
num i/o sched (numsched) : 00000004
disk quota wait (dqwait) : 00000000
1st free ame entry (amefree) : 0000000A
1st del pending ame (ameunxmem) : 00000000
highest ame entry (hiame) : 00000040
pag space free wait (pgspwait) : 00000000
first free apt entry (aptfree) : 00000012
apt high water mark (hiapt) : 0000FFFF
next apt entry (aptlru) : 00000000
first free esid (esidfree) : 00200054
high index of esid (hiesid) : 00001600
first lpgg rsvd sidx (sidxlimit) : 00200000
log high watermark (logmax) : 00000002
sid index of logs (logsidx) : 00476734
lru creation thread (lruwait) : 00000000
memp needing daemon (mempnew) : 00000000
minperm percent (minperm) : 20.0 %
maxperm percent (maxperm) : 80.0 %
maxclient percent (maxclient) : 80.0 %
frame thresholds (minfree, maxfree)
computational : 00000078 00000080
client : 00000078 00000080
persistent : 00000078 00000080
fixed lmb freelist (fixlmbfree) : 00000001
fixed lmb size(pages)(fixlmbsz) : 00000000
fixed lmb firstnfr (fixlmbfirst) : 00000001
fixed lmb lastnfr (fixlmblast) : 00000001
vmpool being deleted (vmp_del) : 00000000
mempool being deleted (memp_del) : 00000000
frameset being deleted (frs_del) : 00000000
global vmap lock @ B0476100 00000000
global ane lock @ B0476180 00000000
global rpt lock @ B0476200 00000000
rpt pool lock [00] @ B0476280 00000000
rpt pool lock [01] @ B0476284 00000000
rpt pool lock [02] @ B0476288 00000000
rpt pool lock [03] @ B047628C 00000000
rpt pool lock [04] @ B0476290 00000000
rpt pool lock [05] @ B0476294 00000000
rpt pool lock [06] @ B0476298 00000000
rpt pool lock [07] @ B047629C 00000000
rpt pool lock [08] @ B04762A0 00000000
rpt pool lock [09] @ B04762A4 00000000
rpt pool lock [10] @ B04762A8 00000000
rpt pool lock [11] @ B04762AC 00000000
rpt pool lock [12] @ B04762B0 00000000
rpt pool lock [13] @ B04762B4 00000000
rpt pool lock [14] @ B04762B8 00000000
rpt pool lock [15] @ B04762BC 00000000
global alloc lock @ B0476300 00000000
apt freelist lock @ B0476380 00000000
pdt allocation lock @ B0476400 00000000
pdt io list lock @ B0476480 00000000
compression page lock @ B0476500 00000000
serv frame alloc lock @ B0476580 00000000
fixlmb freelist lock @ B0476600 00000000
KDB(0)>
pft subcommand

Purpose
The pft subcommand displays information about the VMM page frame table.

Syntax
pft [menu options]

Parameters

menu options Use menu options and parameters with the subcommand to avoid display of menus and prompts.

If the pft subcommand is invoked without parameters, then menus and prompts determine which data is displayed. If the menu selections and required values are known, you can enter them as subcommand parameters.

Aliases
No aliases.

Example
The following is an example of how to use the pft subcommand:

KDB(0)> pft
VMM PFT
Select the PFT entry to display by:
1) page frame #
2) h/w hash (sid,pno)
3) s/w hash (sid,pno)
4) search on swbits
5) search on pincount
6) search for hidden pages
7) scb list
8) io list
9) deferred pgsp service frames
a) scb list (compact output)
b) ksp list (compact output)
Enter your choice: 1
Enter the page frame number (in hex): FC

VMM PFT Entry For Page Frame 000000FC6 of 00000FFFFF

pte = 0000000095F9700  pvt = 0000000000C03F34 pft = 00000000023B40C
h/w hashed sid : 0000000024012 pno : 000000FF3C key : 0
source sid : 00024012 pno : 000000FF3C key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id (vmp) : 0000
owning mempool id (memp) : 0000
owning frameset (frs) : 0001
page number in scb (spage) : FF3C
disk block number (dblock) : 00000000
next page on scb list (sidfwd) : 000000FC6
prev page on scb list (sidbwd) : 0005F6D4
freefwd/waitlist (freefwd) : 00000000
freebwd/logage/pincnt (freebwd) : 00010000
out-of-order I/O (nonfifo): 00000000
(0)> more (^C to quit) ?
storage attributes (wimg) : 2
next page on s/w hash (next) : FFFFFFFF
List of alias entries (alist) : 0000FFFF
index in PDT (devid) : 0000
next frame i/o list (nextio) : 00000000
save key across pagein(savekey): 0
KDB(0)> pft 2
Enter the sid (in hex): 24012
Enter the pno (in hex): FF3C
VMM PFT Entry For Page Frame 0000000FCDO of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 000000000203B40C
h/w hashed sid : 0000000024012 pno : 000000FF3C key : 0
source sid : 00024012 pno : 000000FF3C key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id (vmp) : 0000
owning mempool id (memp) : 0000
owning frameset (frs) : 0001
page number in scb (spage) : FF3C
disk block number (dblock) : 00000000
next page on scb list (sidfwd) : 00000FC6
prev page on scb list (sidbwd) : 0005F6D4
freefwd/waitlist (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00010000
out-of-order I/O (nonfifo): 00000000
(0)> more (^C to quit) ?
storage attributes (wimg) : 2
next page on s/w hash (next) : FFFFFFFF
List of alias entries (alist) : 0000FFFF
index in PDT (devid) : 0000
next frame i/o list (nextio) : 00000000
save key across pagein(savekey): 0
KDB(0)> pft 3 24012 FF3C
VMM PFT Entry For Page Frame 0000000FCDO of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 000000000203B40C
h/w hashed sid : 0000000024012 pno : 000000FF3C key : 0
source sid : 00024012 pno : 000000FF3C key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id (vmp) : 0000
owning mempool id (memp) : 0000
owning frameset (frs) : 0001
page number in scb (spage) : FF3C
disk block number (dblock) : 00000000
next page on scb list (sidfwd) : 00000FC6
prev page on scb list (sidbwd) : 0005F6D4
freefwd/waitlist (freefwd): 00000000
freebwd/logage/pincnt (freebwd): 00010000
out-of-order I/O (nonfifo): 00000000
(0)> more (^C to quit) ?
storage attributes (wimg) : 2
next page on s/w hash (next) : FFFFFFFF
List of alias entries (alist) : 0000FFFF
index in PDT (devid): 0000
next frame i/o list (nextio): 00000000
save key across pagein(savekey): 0
KDB(0)> pft 7
Enter the sid (in hex): 00024012
VMM PFT Entry For Page Frame 0000000FCF of 000005FFFF

pte = 00000000095F8700 pvt = 0000000000C03F3C pft = 0000000000203B484
h/w hashed sid : 00000000024012 pno : 000000FF7C key : 0
source sid : 00024012 pno : 000000FF7C key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/1
owning vmpool id (vmp) : 0000
owning mempool id (memp) : 0000
owning frameset (frs) : 0001
page number in scb (spage) : FF7C
disk block number (dblock) : 00000000
next page on scb list (sidfwd) : 000005F6D4
prev page on scb list (sidbwd) : FFFFFFFF
freefwd/waitlist (freefwd) : 00000000
freebwd/logage/pincnt (freebwd) : 00000000
out-of-order I/O (nonfifo) : 00000000
(0)> more (^C to quit) ?
storage attributes (wimg) : 2
List of alias entries (alist) : 0000FFFF
index in PDT (devid) : 0000
next frame i/o list (nextio) : 00000000
save key across pagein(savekey): 0

VMM PFT Entry For Page Frame 000005F6D4 of 000005FFFF

pte = 00000000095F9400 pvt = 0000000000D7DB50 pft = 000000000365D9B0
h/w hashed sid : 0000000024012 pno : 000000FF3A key : 0
source sid : 00024012 pno : 000000FF3A key : 0

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/1
> modified (pft/pvt/pte): 1/0/0
owning vmpool id (vmp) : 0000
owning mempool id (memp) : 0000
owning frameset (frs) : 0000
page number in scb (spage) : FF3A
(0)> more (^C to quit) ?
disk block number (dblock) : 00000000
next page on s/w hash (next) : FFFFFFFF
prev page on scb list (sidfwd) : 000005F6D4
prev page on scb list (sidbwd) : 00000000
freefwd/waitlist (freefwd) : 00000000
freebwd/logage/pincnt (freebwd) : 00000000
out-of-order I/O (nonfifo) : 00000000
storage attributes (wimg) : 2
List of alias entries (alist) : 0000FFFF
index in PDT (devid) : 0000
next frame i/o list (nextio) : 00000000
save key across pagein(savekey): 0

VMM PFT Entry For Page Frame 0000000FCF of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 0000000000203B484

VMM PFT Entry For Page Frame 000005F6D4 of 000005FFFF

pte = 00000000095F9400 pvt = 0000000000D7DB50 pft = 000000000365D9B0
h/w hashed sid : 0000000024012 pno : 000000FF3A key : 0
source sid : 00024012 pno : 000000FF3A key : 0

(0)> more (^C to quit) ?
disk block number (dblock) : 00000000
next page on s/w hash (next) : FFFFFFFF
prev page on scb list (sidfwd) : 000005F6D4
prev page on scb list (sidbwd) : 00000000
freefwd/waitlist (freefwd) : 00000000
freebwd/logage/pincnt (freebwd) : 00000000
out-of-order I/O (nonfifo) : 00000000
storage attributes (wimg) : 2
List of alias entries (alist) : 0000FFFF
index in PDT (devid) : 0000
next frame i/o list (nextio) : 00000000
save key across pagein(savekey): 0

VMM PFT Entry For Page Frame 000005F6D4 of 000005FFFF

pte = 00000000095F9700 pvt = 0000000000C03F34 pft = 0000000000203B40C
h/w hashed sid : 0000000024012  pno : 000000FF3C  key : 0
source sid : 00024012  pno : 000000FF3C  key : 0

> in use
> on scb list
> valid (h/w)
(0)= more (^C to quit) ?

KDB(0)> pft a
Enter the sid (in hex): 00024012
Frame  Ord..page Pincount Dblock Key ...
00000FCF  FF7C  00000000  00000000  K MOD REF
0005F6D4  FF3A  00010000  00000000  K MOD REF
00000FCD  FF3C  00010000  00000000  K MOD REF
00000FC6  FF3B  00020000  00000000  K MOD REF

Pages on SCB list
npages.......... 00000004
on sidlist....... 00000004
file pageout.... 00000000
pageout_pagein.. 00000000
KDB(0)> pft 5

Page frames with pincount > 0:
00000, 00002-005A3, 006F0-006F4, 0082D-008FF, 00C0E-00C10
00C20-00C27, 00D0-00D07, 00D8, 00FB4, 00FB6-00FB8, 00FB8-00FC7
00FCA-00FCE, 00FD0-00FD2, 00FD4-00FD9, 00FDB, 00FDD, 00FE0-00FFF
01007, 01017, 01019, 0102C, 01033, 01038
0103A, 0103C, 0103E, 01040, 01042-01044, 01046
01048, 0104F, 01051, 01053, 01055, 01057
01059, 0105B, 0105D, 0105F, 01065, 01084
01086, 01088, 0108A, 0108C, 0108E, 010C0
010C2, 010C4, 010CC, 010CE-010D1, 010D3, 010D5
010D7, 010D9, 010DB, 010D0, 010DF, 010E3
010E9, 010ED, 010EF, 010F1, 01160
0116A, 0116C, 0116E, 01170, 01172, 01174
01176, 01178, 0117A, 0117C, 0117E, 01180
01182-01184, 01186, 01188, 0118A, 0118C, 0118E
01190, 01192, 01194, 01196-01197, 01199, 0119B, 0119D
0133D, 0133F, 01341, 01343, 01345, 01347
01349, 0134B, 0134D, 0134F, 01351, 01353
01355, 01357, 01359, 0135B, 0135D, 0135F
01361, 01363-01364, 01366, 01368, 0136A, 0136C
0136E, 01370, 01372, 01374, 01376, 01378-0137A

Chapter 24. Display VMM subcommands 193
**swat subcommand**

**Purpose**
The `swat` subcommand displays VMM SW hash table entries. It can also be used to look for corrupted SW hash table entries.

**Syntax**
```
swat [1..3]
swat 1 [index]
swat 2 [sid pno]
swat 3
```

**Parameters**
- `index` Indicates the `swat` index.
- `sid` Indicates the virtual segment identifier.
- `pno` Indicates the page number.

When the `swat` subcommand is given no parameters, a menu is displayed with the following options:

1. Displays the software hash table entry identified by a `swat` index entered by the user.
2. Displays the software hash table entry identified by a `sid` (virtual segment identifier) and `pno` (page number) entered by the user.
3. Checks for corruption in the `swat` by examining the stored page frame numbers.

The command completes after it runs one of the options. To exit the menu and terminate the command without running any of the options, enter a period (`.`).

**Note:** You can enter multiple parameters simultaneously.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `swat` subcommand:
```
KDB(0)> swat
VMM SWAT
Select the SWAT option:
1) display by index
2) hash by (sid,pno)
3) look for invalid entries
Enter your choice: 1
Enter the swat index (in hex): 88
vmmswat+000220 swat[00000088]: 00000088
KDB(0)> swat 1 88
vmmswat+000220 swat[00000088]: 00000088
KDB(0)> swat 2
Enter the sid (in hex): 0
Enter the pno (in hex): 88
vmmswat+000220 swat[00000088]: 00000088
```
KDB(0)> swhat 3
There are 00000000 corrupt entries.
KDB(0)>
pvt subcommand

Purpose
The pvt subcommand displays the VMM PVT and PVLIST entries. The pvt subcommand can also be
used to look for corrupted PVT and PVLIST entries.

Syntax
pvt [1..4]
pvt 1 [index]
pvt 2
pvt 3 [index]
pvt 4

Parameters
index Identifies the PVT or PVLIST index for which you want PVT information.

If you use the pvt subcommand with no parameters, a menu with four options is displayed. Choose one of
the options, or type the parameters with the options as part of the subcommand. The options you can
choose or type are the following:

1  Displays the PVT identified by a PVT index entered by the user.
2  Checks the PVT entry for every page with a valid software pft entry by examining the pte index
   stored in the PVT. Entries identified as corrupted are printed.
3  Displays the PVLIST identified by a PVLIST index entered by the user.
4  Checks the PVLIST entry for each pte index by examining the pvnext field in the PVLIST.

The subcommand terminates after running one of the options.

To exit the menu and terminate the subcommand without running any of the options, enter a period (.)

Note: Multiple parameters can be entered simultaneously.

Aliases
pplist

Example
The following is an example of how to use the pvt subcommand:

KDB(0)> pvt
VMM PVT/PVLIST
Select the PVT/PVLIST option:
  1) display pvt by index
  2) look for invalid pvt entries
  3) display pvlist by index
  4) look for invalid pvlist entries
Enter your choice: 1
Enter the pvt index (in hex): 88
   NFR     PTEX    REF MOD     RAW BITS
p64pvt+000220 0000000888 00000440 0 0 00000440
KDB(0)> pvt 1 88

196  KDB Kernel debugger and kdb command
p64pvt+000220  0000000088  00000440  0  0  00000440
KDB(0)> pvt 3
Enter the pvlist index (in hex): 440
INDEX   PNO   NEXT   RAW_BITS
00000440  088   3FFFFFFF  000000883FFFFFFF
KDB(0)> pvt 2
There are 00000000 corrupt entries.
KDB(0)> pvt 4
There are 00000000 corrupt entries.
KDB(0)>
pta subcommand

Purpose
The pta subcommand displays data from the VMM PTA segment.

Syntax

Parameters
-\( r \) Displays XPT root data.
-\( d \) Displays XPT direct block data.
-\( a \) Displays Area Page Maps or a specific Area Page Map.
-\( v \) Displays map blocks.
-\( x \) Displays XPT fields.
-\( f \) Prompts for the \( sid \) or \( idx \) for which the XPT fields are to be displayed.
\( sid \) Specifies the segment ID. Symbols, hexadecimal values, or hexadecimal expressions may be used for this argument.
\( idx \) Specifies the index for the specified area. Symbols, hexadecimal values, or hexadecimal expressions can be used for this argument.

The optional arguments listed above determine the data that is displayed. Summary information is displayed when no parameter is provided.

Aliases
No aliases.

Example
The following is an example of how to use the pta subcommand:

KDB(0)\> pta -?
VMM PTA segment (1) @ C0000000
Usage: pta
pta -[r] [root] [sid] [seg no.] /to print XPT root
pta -[d] [blk] [sid] [seg no.] /to print XPT direct blocks
pta -[a] [pm] [idx] [seg no.] /to print Area Page Maps
pta -[apm] [apmidx] [segno] /to print specific APM
pta -v [map] [idx] [seg no.] /to print map blocks
pta -[x] [pt] xpt /to print XPT fields
pta -f [ind] (prompt for sid/pno) /to find or print XPT fields
KDB(0)\> pta
VMM PTA segment (1) @ C0000000
VMM PTA segment @ C0000000
pta_root..... @ C0000000 pta_hiapm.... : 00000200
pta_vmapfree... : 00000000 pta_usecount.... : 00004000
pta_anchor(0).... : 00000000 pta_anchor(1).... : 00000000
pta_anchor(2).... : 00000000 pta_anchor(3).... : 00000000
pta_anchor(4).... : 00000000 pta_anchor(5).... : 00000000
pta_freecnt.... : 00000000 pta_freetail.... : 000001FF
pta_apm(1rst).... @ C0000600 pta_xptdblk.... @ C0080000
KDB(0)\>
pte subcommand

Purpose
The pte subcommand provides options for displaying information about the VMM page table entries.

Syntax
pte [menu options]

Parameters
menu options
Use menu options and parameters with the subcommand to avoid the display of menus and prompts.

If the pte subcommand is invoked without parameters, menus and prompts are used to determine the data to be displayed. If the menu selections and required values are known, you can use them as subcommand parameters.

Aliases
No aliases.

Example
The following is an example of how to use the pte subcommand:

KDB(0)> pte
VMM PTE
Select the PTE to display by:
1) index
2) sid,pno
3) page frame
4) PTE group
Enter your choice: 2
Enter the sid (in hex): 400
Enter the pno (in hex): 0

PTEX v SID h avpi RPN r c wimg pp L pin
002001 1 0000000000400 0 00 0000000021E36 1 0 0002 01 0 0

KDB(0)> pte 4
Enter the sid (in hex): 400
Enter the pno (in hex): 0

PTEX v SID h avpi RPN r c wimg pp L pin
002000 1 0000000000000 0 00 0000000000400 1 0 0002 00 0 0
002001 1 0000000000400 0 00 0000000021E36 1 0 0002 01 0 0
002002 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
002003 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
002004 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
002005 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
002006 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
002007 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
01FDFF8 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
01FDFF9 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
01FDFFA 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
01FDFFB 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
01FDFFC 0 0000000000000 0 00 0000000000000 0 0000 00 0 0
01FDFFE 0 0000000000000 0 00 0000000000000 0 0000 00 0 0

Chapter 24. Display VMM subcommands 199
1FDFF 0 000000000000 0 0 0 0 0 0 0 0 0 0 0 0 0

KDB(0)>
rmap subcommand

Purpose
The rmap subcommand displays the real address range mapping table.

Syntax
rmap [*] [slot]

Parameters
- Displays all real address range mappings.
slot Displays the real address range mapping for the specified slot. This value must be a hexadecimal value.

If the asterisk (*) parameter is specified, a summary of all entries is displayed. If a slot number is specified, only that entry is displayed. If no parameter is specified, the user is prompted for a slot number, and data for that and all higher slots is displayed.

Aliases
No aliases.

Example
The following is an example of how to use the rmap subcommand:

KDB(0)> rmap *

<table>
<thead>
<tr>
<th>SLOT</th>
<th>RADDR</th>
<th>SIZE</th>
<th>V</th>
<th>ALIGN</th>
<th>&lt;name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmrmap+000030 01 00000000000000093534F 0 00000000 Kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000058 02 00007FAC0000000008FEC 0 00000000 IPL control block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000080 03 00000936000000021000 0 00001000 MST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000120 07 00002000000016B0000 0 00400000 s/w PFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000148 0A 00000C00000000018000 0 00400000 PVT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000170 09 000003680000000100000 0 00001000 PVLIST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000198 0A 0000080000000002000000 0 02000000 PFT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+0001C0 0B 000009570000000100000 0 00001000 s/w HAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+0001E8 0C 00000A570000000100000 0 00001000 APT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000210 0D 00000B570000000200000 0 00001000 ANAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000238 0E 00000B770000000800000 0 00001000 RPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000260 0F 00000D800000000200000 0 00001000 RPHAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000288 10 00000A00000000018000 0 00001000 PSH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+0002B0 11 00000B700000000100000 0 00001000 PTD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+0002C0 12 00000BF800000000200000 0 00001000 PVD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000300 13 00000BFA000000030000 0 00001000 PTAI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000328 14 00000BF200000000100000 0 00001000 DMAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000340 15 00000B880000000200000 0 00001000 MEM_POOL &amp; FRAME_SET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000468 1C 00000FE20000001E000 0 00000000 RMAILOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vmrmap+000490 1D 00000FE000000020000 0 00001000 VMINT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KDB(0)> rmap 11
RMAP entry 0011 of 004F: PTAR
> valid
> has mempool/frameset ids
Real address : 0000000000000000BF7000
Effective address : 0000000000000000C0000000
Size : 00000000000000010000
Alignment : 00000000000000010000
WIMG bits : 2
vmpool requested : 00
vmpool actual : 00

Chapter 24. Display VMM subcommands
KDB(0) > rmap
VMM usage: rmap [*][slot]
Enter the RMAP index (0-004F): 11
RMAP entry 0011 of 004F: PTAR
> valid
> has mempool/frameset ids
Real address : 00000000000BF7000
Effective address : 00000000000C000000
Size : 000000000000001000
Alignment : 000000000000001000
WIMG bits : 2
vmpool requested : 00
vmpool actual : 00
RMAP entry 0012 of 004F: PTAD
> valid
> has mempool/frameset ids
Real address : 00000000000BF8000
Effective address : 00000000000C000000
Size : 000000000000002000
Alignment : 000000000000001000
WIMG bits : 2
vmpool requested : 00
vmpool actual : 00
RMAP entry 0013 of 004F: PTAI
> valid
(0) > more (^C to quit)?
> has mempool/frameset ids
Real address : 00000000000BFA000
Effective address : 00000000000D000000
Size : 000000000000003000
Alignment : 000000000000001000
WIMG bits : 2
vmpool requested : 00
vmpool actual : 00
RMAP entry 0014 of 004F: DMAP
> valid
> has mempool/frameset ids
Real address : 00000000000BFD000
Effective address : 00000000000D000000
Size : 000000000000001000
Alignment : 000000000000001000
WIMG bits : 2
vmpool requested : 00
vmpool actual : 00
RMAP entry 0015 of 004F: unknown
RMAP entry 0016 of 004F: unknown
<snip>
rvsid subcommand

Purpose
The \texttt{rvsid} subcommand displays reserved vsid information (struct rvsid_data).

Note: The \texttt{rvsid} subcommand is only supported when you use the \texttt{kdb} command or the KDB kernel debugger on the 64-bit kernel.

Syntax
\texttt{rvsid}

Parameters
No parameters are supported.

Aliases
No aliases.

Example
The following is an example of how to use the \texttt{rvsid} subcommand:

\begin{verbatim}
(0)> rvsid

Reserved Vsid Control Variables: 000000000023D4E0rvsid_da+000000

num lgpg vsids per group (lgpg_vsids_per_group) : 00000006
use spec. lgpg vsid alloc (lgpg_vsid_on) : 00000000
rsvd vsid alloc interval (sid_int) : 00000200
number of reserved vsids (num_vsids) : 00000000
highest reserved vsid (hi_vsid) : 00000000
highest reserved sidx+1 (hi_sidx) : 00000000
num reserved vsids in use (num_inuse) : 00000000
reserved vsids high water (hi_inuse) : 00000000
(0)>
\end{verbatim}
**scb subcommand**

**Purpose**
The `scb` subcommand provides options for display of information about VMM segment control blocks.

**Syntax**
`scb [menu options]`

**Parameters**
*menu options* Use menu options and parameters with the subcommand to avoid display of menus and prompts.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data that is displayed. If the menu selections and required values are known, you can use them as subcommand parameters.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `scb` subcommand:

```
KDB(0)> scb
VMM SCBs
Select the scb to display by:
  1) index
  2) sid
  3) srval
  4) search on sibits
  5) search on npsblks
  6) search on nvpages
  7) search on npages
  8) search on npseablks
  9) search on ninpages
 10) search on lock
    a) search on segment type
    b) add total scb_vpages
    c) search on segment class
    d) search on segment pvproc
Enter your choice: 2
Enter the sid (in hex): 00024012
```

```
VMM SCB Addr B04775F4 Index 00000012 of 0000050B  Segment ID: 00024012

WORKING STORAGE SEGMENT
> (_segtype)..... working segment
> (_defd)......... deferred disk alloc
> (_privseg)..... process private segment
> (_compseg)..... computational segment
> (_privatt)..... process attachment
segment info bits  (sibits) : 88408800
default storage key (_defkey) : 2
extent of growing down  (minvpn) : 0000FF3A 65338
last page user region  (sysbr) : FFFFFFFF -1
up limit  (uplim) : 00000000 0
down limit (downlim) : 0000EF23 61219
number of pgsp blocks (npsblks) : 00000000 0
number of virtual pages (vpages) : 00000004 4
freeze count  (frozen) : 00000000 0
```
number of epsa blocks (npseablks): 00000000 0
XPT root seg number (xptrseg) : 00000002 2
offset of XPT root (xptroff) : 00000302 770
XPT root address : C00C0800
(0)> more ("C to quit") ?
class ID (classid) : 00000000 0
physical attachments (_att) : 00000000
mmap reference count (refcnt) : 00000000
pvproc ptr & pvid : E2000400 00000204
mempools : 0000000000000000
non-fblu pageout count (npopages) : 0000
xmem attach count (xmemcnt) : 0000
pages in real memory (npages) : 00000004
pinned pages in memory (npinpages) : 00000003
lru pageout count (npopages) : 00000000
proc pointer (proc) : E2000400
page frame at head (sidlist) : 000000FCF
max assigned page number (maxvpn) : FFFFFFFF
lock (lock) : 00047764C 00000000
KDB(0)> scb
VMM SCBs
Select the scb to display by:
1) index
2) sid
3) srval
4) search on sibits
5) search on npseablks
6) search on npvpages
7) search on npages
8) search on npseablks
9) search on npinpages
10) search on lock
   a) search on segment type
   b) add total scb_vpages
   c) search on segment class
   d) search on segment pvproc
Enter your choice: 7

Find all scbs whose npages is greater than (in hex): 2000

VMM SCB Addr B04774E0 Index 0000000F of 0000050B Segment ID: 0001E00F

WORKING STORAGE SEGMENT
> (_segtype)...... working segment
> (_defd)......... deferred disk alloc
> (_system)...... system segment
> (_compseg)..... computational segment
segment info bits (_sibits) : 88080000
default storage key (_defkey) : 2
extent of growing down (minvnpn) : 00010000 65536
up limit (uplim) : 00000FFF 65535
down limit (downlim) : 00010000 65536
number of pgsp blocks (npseablks) : 00000000 0
number of virtual pages (vpages) : 000030F8 12536
freeze count (frozen) : 00000000 0
number of epsa blocks (npseablks) : 00000000 0
XPT root seg number (xptrseg) : 00000001 1
offset of XPT root (xptroff) : 00000333 819
XPT root address : C00CCC00
class ID (classid) : 00000000 0
physical attachments (_att) : 00000000
(0)> more ("C to quit") ?
mmap reference count (refcnt) : 00000000
non-fblu pageout count (npopages) : 0000
xmem attach count (xmemcnt) : 0015
pages in real memory (npages) : 0000030F8
pinned pages in memory (npinpages) : 000000CD4

Chapter 24. Display VMM subcommands 205
lru pageout count : 00000000
proc pointer : 0028F908
page frame at head : 0005F2E0
max assigned page number : 000038A2
lock : 0B0477538 00000000

00000001 (hex) matches found with npages > 00002000.
KDB(0)> scb 1
Enter the index (in hex): 0000000F

VMM SCB Addr B04774E0 Index 0000000F of 0000050B Segment ID: 0001E00F

WORKING STORAGE SEGMENT
> (_segtype)..... working segment
> (_defd)......... deferred disk alloc
> (_system)....... system segment
> (_compseg)..... computational segment
segment info bits : 88088000
default storage key : 2
extent of growing down : 00010000 65536
up limit : 0000FFFF 65535
down limit : 00010000 65536
number of pgsp blocks : 00000000 0
number of virtual pages : 000030F8 12536
freeze count : 00000000 0
number of epsa blocks : 00000000 0
XPT root seg number : 00000001 1
offset of XPT root : 00000333 819
XPT root address : C00CCC00
class ID : 00000000 0
physical attachments : 00000000
(0)> more (^C to quit) ?
mmap reference count : 00000000
non-fblu pageout count : 0000
xmem attach count : 0015
pages in real memory : 0000030F8
pinned pages in memory : 000000C04
lru pageout count : 00000000
proc pointer : 0028F908
page frame at head : 0005F2E0
max assigned page number : 000038A2
lock : 0B0477538 00000000
KDB(0)>
segst64 subcommand

Purpose
The segst64 subcommand displays the segment state information for a 64-bit process.

Syntax
segst64 [-p pid | -e esid | -s seg | value]

Parameters
- **-p pid**  Specifies the process ID of a 64-bit process. This must be a decimal or hexadecimal value depending on the setting of the hexadecimal_wanted switch.
- **-e esid**  Specifies the first segment register to display. The lower register numbers 0, 1, and 2 are ignored. This parameter must be a hexadecimal value.
- **-s seg**  Limits the display to only segment register with a segment state that matches seg. Possible values for seg are: SEG_AVAIL, SEG_SHARED, SEG_MAPPED, SEG_MRDWR, SEG_DEFER, SEG_MMAP, SEG_WORKING, SEG_RMMAP, SEG_OTHER, SEG_EXTSHM, and SEG_TEXT.
- **value**  Sets the limit to display only segments with the specified value for the segfileno field. This value must be hexadecimal.

Aliases
No aliases.

Example
The following is an example of how to use the segst64 subcommand:

```
KDB(0)> segst64 //display
snode base last nvalid sfwd sbwd
00000000 00000003 FFFFFFFE 00000010 00000001 FFFFFFFF
ESID segstate segflag num_segs fno/shmp/srval/nsegs
SR00000003>[ 0] SEG_AVAIL 00000000 0000000A
SR0000000D>[ 1] SEG_OTHER 00000001 00000001
SR0000000E>[ 2] SEG_AVAIL 00000000 00000001
SR0000000F>[ 3] SEG_OTHER 00000001 00000001
SR00000010>[ 4] SEG_TEXT 00000001 00000001
SR00000011>[ 5] SEG_WORKING 00000001 00000000
SR00000012>[ 6] SEG_AVAIL 00000000 8000FFFF
SR00000014>[ 7] SEG_WORKING 00000001 00000000
SR00000015>[ 8] SEG_AVAIL 00000000 00010009
SR80020015>[10] SEG_AVAIL 00000000 0FFDFFEA
SR90000000>[12] SEG_TEXT 00000001 00000001
SR90000001>[13] SEG_AVAIL 00000000 0FFFFFFE
SRFFFFFFFF>[14] SEG_TEXT 00000001 00000001
SRFFFFFFFF>[15] SEG_AVAIL 00000000 0FFFFFFF
```

Chapter 24. Display VMM subcommands  207
sr64 subcommand

Purpose
The sr64 subcommand displays segment registers for a 64-bit process for 32-bit and 64-bit kernels.

Syntax

sr64 [-p pid] [esid] [size] // for 32 bit kernel

sr64 [-g range size] [-p pid] [esid] [size] // for 64 bit kernel

Parameters

-p pid  Specifies the process ID of a 64-bit process. This must be a decimal or hexadecimal value, depending on the setting of the hexadecimal_wanted switch. The hexadecimal_wanted switch is changed using the set subcommand.

-g     Displays esids from the global system address space. The minimum range size and the default is 3, but a larger range can optionally be provided.

   Note: The -g flag is only supported for the 64-bit kernel.

esid   Specifies the first segment register to display. Register numbers lower than the specified register are ignored. This parameter must be a hexadecimal value.

size   Specifies the value to be added to the first segment register to determine the last segment register to display. This parameter must be a hexadecimal value.

If no parameters are specified, the current process is used. Another process can be specified by using the -p pid flag. Additionally, the esid and size parameters can be used to limit the segment registers displayed. The esid value determines the first segment register to display. The value of esid + size determines the last segment register to display.

The registers are displayed in groups of 16. If necessary, the value of the esid parameter is rounded down to a multiple of 16, and the size is rounded up to a multiple of 16. For example: sr64 11 11 displays the segment registers 10 through 2f.

Aliases
No aliases.

Example
The following is an example of how to use the sr64 subcommand for a 32-bit kernel:

KDB(0)> sr64 ?  //display help
Usage: sr64 [-p pid] [esid] [size]
KDB(0)> sr64 11 11 //display up to 16 SRs from 10
Segment registers for address space of Pid: 000048CA
SR00000000: 60000000  SR00000002: 60002B45  SR00000004: 6000614C
SR00000010: 6000520A  SR00000011: 6000636C
SR8001000A: 60003B47
SR80020014: 6000B356
SRBFFFFFFFF: 6000340
SR90000000: 60001142
SRBFFFFFFFF: 60004148
SRBFFFFFFFF: 60008336
KDB(0)> sr64 0 100  //display up to 256 SRs from 0
Segment registers for address space of Pid: 000048CA
SR00000000: 60000000  SR00000002: 60002B45  SR00000004: 6000614C
SR00000010: 6000520A  SR00000011: 6000636C
The following is an example of how to use the `sr64` subcommand for a 64-bit kernel:

```
KDB(0)> sr64 -g
Segment registers for global address space
kernel..... sr000000000: 0000400
vmm data... srF100000004: 00801400
vmm pta.... srF100000005: 01002400
vmm diskmap srF100000006: 01803400 srF100000007: 02004400 srF100000008: 02805400 ..
vmm ame.... srF10000000A: 03807400 srF10000000B: 04008400 srF10000000C: 04809400 ..
vmm scb.... srF10000000E: 0580B400 srF10000000F: 0600C400 srF100000010: 0680D400 ..
vmm swat.. srF10000000E: 0688B400 srF10000000F: 0E08C400 srF10000000C: 0E8BD400 ..
real heap.. srF100000013E: 0E930400 srF100000013F: 0E13C400 srF1000000140: 0E930400 ..
proc-thread srF10000007B: 0A075400 srF100000079: 0B876400 srF100000087A: 0B077400 ..
mbuf....... srF10000008C: 0C099400 srF10000008D: 0D89A400 srF10000008E: 0009B400 ..
l dr........ srF1000000A0: 0E19D400 srF1000000A1: 0F99E400 srF1000000A2: 0F19F400 ..
jfs l kword. srF1000000C0: 0E1BD400
kernel heap srF1000000F0: 0E6FD400 srF1000000F1: 0FEFE400 srF1000000F2: 0F6FF400 ..
global ext. srF1000000F2: 071EF400 srF1000000F3: 089F0400 srF1000000F4: 081F1400 ..
global l gpg srF1000000E0: 066D400 srF1000000E1: 0FEDE400 srF1000000E2: 0F6DF400 ..
vmm ksp.... srF20001001: 1001001004 srF20001002: 1020002400 srF20001003: 1030003400 ..
KDB(0)> sr64 -g 6
Segment registers for global address space
kernel..... sr000000000: 0000400
vmm data... srF100000004: 00801400
vmm pta.... srF100000005: 01002400
vmm diskmap srF100000006: 01803400 srF100000007: 02004400 srF100000008: 02805400 ..
vmm ame.... srF10000000A: 03807400 srF10000000B: 04008400 srF10000000C: 04809400 ..
vmm scb.... srF10000000E: 0580B400 srF10000000F: 0600C400 srF100000010: 0680D400 ..
vmm swat.. srF10000000E: 0688B400 srF10000000F: 0E08C400 srF10000000C: 0E8BD400 ..
vmm swat.. srF10000000C1: 0F8BE400 srF10000000C2: 0F8BF400 srF10000000C3: 00000400 ..
real heap.. srF100000013E: 0E930400 srF100000013F: 0E13C400 srF1000000140: 0E930400 ..
real heap.. srF1000000141: 0F13E400
proc-thread srF100000078: 0A075400 srF100000079: 0B876400 srF100000087A: 0B077400 ..
proc-thread srF100000078: 0C87B400 srF10000007C: 0C079400 srF100000087D: 0D87A400 ..
mbuf....... srF10000008C: 0C099400 srF10000008D: 0D89A400 srF10000008E: 0009B400 ..
mbuf....... srF10000008F: 0E93E400 srF10000008A0: 0E09D400 srF10000008A1: 0F99E400 ..
l dr........ srF1000000A0: 0E19D400 srF1000000A1: 0F99E400 srF1000000A2: 0F19F400 ..
l dr........ srF1000000A3: 009A400 srF1000000A4: 001A1400 srF1000000A5: 019A2400 ..
jfs l kword. srF1000000C0: 0E1BD400
kernel heap srF1000000F0: 0E6FD400 srF1000000F1: 0FEFE400 srF1000000F2: 0F6FF400 ..
kern el heap srF1000000F3: 00F00400 srF1000000F4: 0701400 srF1000000F5: 01F02400 ..
global ext. srF1000000F2: 071EF400 srF1000000F3: 089F0400 srF1000000F4: 081F1400 ..
global ext. srF1000000F5: 099F2400 srF1000000F6: 091F3400 srF1000000F7: 0A0F4400 ..
global l gpg srF1000000E0: 066D400 srF1000000E1: 0FEDE400 srF1000000E2: 0F6DF400 ..
global l gpg srF1000000E3: 00EE0400 srF1000000E4: 006E1400 srF1000000E5: 01EE2400 ..
```
vmm ksp.... srF20001001: 10010001400 srF20001002: 10020002400 srF20001003: 10030003400
vmm ksp.... srF20001004: 10040004400 srF20001005: 10050005400 srF20001006: 10060006400 ..
KDB(0)>
**ksp subcommand**

**Purpose**
The `ksp` subcommand displays information about the Kernel Special Purpose (KSP) region.

**Note:** Because some of the contents of the KSP region depend upon whether you are using the 32-bit or 64-bit kernel, the output of the `ksp` subcommand varies slightly depending upon which kernel you use.

**Syntax**
```
  ksp
```

**Parameters**
No parameters.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `ksp` subcommand:

```
KDB(0)> ksp

Kernel Special Purpose (KSP) Region Info

KSP_FIRST_SID........010000000
KSP_SID_BASE.........010010001
KSP_SIDX_BASE........010010001
KSP_SIDHASH_INC.......00010001
KSP_REGION_INC.......010000000
KSP_SID_END..........02D830D83
KSP_ESID_BASE........F20001001
KSP_ESID_END.........F20003000
KSP_TOTAL_SIDS........00000D82
KSP_ARCH_NUMSIDS......00000D82

Data Structures in the KSP Region:

VMM SWPFT Address............F2000100100000000 vmmswpft+000000
VMM SWPFT Esid Range........F200010001, F20001001
VMM SWPFT Start (sidx,sid)...010010001, 010010001
VMM SWPFT End (sidx,sid).....010010001, 010010001
VMM SWPFT Size in #Segments..(partial segment)

VMM HWPFT Address............0000000000000000
VMM HWPFT Esid Range........0000000000, 0000000000
VMM HWPFT Start (sidx,sid)...0000000000000000
VMM HWPFT End (sidx,sid).....0000000000000000
VMM HWPFT Size in #Segments..00000001

VMM PVT Address............F2000100200000000
VMM PVT Esid Range..........F20001002, F20001002
VMM PVT Start (sidx,sid)...010020002, 010020002
VMM PVT End (sidx,sid).....010020002, 010020002
VMM PVT Size in #Segments..(partial segment)

VMM PVLIST Address.........F2000200300000000
VMM PVLIST Esid Range.......F20002003, F20002003
VMM PVLIST Start (sidx,sid)...020030003, 020030003
VMM PVLIST End (sidx,sid)....020030003, 020030003
```
VMM PVLIST Size in #Segments..(partial segment)

Segment ID and related definitions for reference

NUMSIDS..............10000000
VM_L2_MAXARCH_VSID....00000025
VM_MAXARCH_VSID.......1FFFFFFFFF
VM_L2_IOSID_BIT......00000024
IOSIOBIT...............1000000000
IOSIDMASK.............FFFFFFFFF
GLOB_ESID_LAST........F10000FFF

(0)>
ste subcommand

Purpose
The ste subcommand provides options for displaying information about segment table entries for 64-bit processes.

Syntax
ste [-p pid] [menu options]

Parameters
-p pid Specifies the process identifier to switch to before the menu is invoked. If this optional flag is omitted, the current process is assumed.
menu options Enter menu options and parameters along with the subcommand to avoid displaying menus and prompts. If you do not enter menu options, the menu is invoked.

If this subcommand is invoked without parameters, then menus and prompts are used to determine the data to display.

Aliases
No aliases.

Example
The following is an example of how to use the ste subcommand:
KDB(0)> ste -p 042B8
Switch to proc: E2008400
Segment Table (STAB)
Select the STAB entry to display by:
1) esid
2) sid
3) dump hash class (input=esid)
4) dump entire stab
Enter your choice: 4
0000000022821000: ESID 0000000009000000 VSID 0000000000041A2 V Ks Kp
0000000022821010: ESID 0000000000000000 VSID 0000000000000000 V Ks Kp
0000000022821020: ESID 0000000000000000 VSID 0000000000000000
0000000022821030: ESID 0000000000000000 VSID 0000000000000000
0000000022821040: ESID 0000000000000000 VSID 0000000000000000
0000000022821050: ESID 0000000000000000 VSID 0000000000000000
0000000022821060: ESID 0000000000000000 VSID 0000000000000000
0000000022821070: ESID 0000000000000000 VSID 0000000000000000
0000000022821080: ESID 0000000000000000 VSID 0000000000000000
0000000022821090: ESID 0000000000000000 VSID 0000000000000000
00000000228210A0: ESID 0000000000000000 VSID 0000000000000000
00000000228210B0: ESID 0000000000000000 VSID 0000000000000000
00000000228210C0: ESID 0000000000000000 VSID 0000000000000000
00000000228210D0: ESID 0000000000000000 VSID 0000000000000000
00000000228210E0: ESID 0000000000000000 VSID 0000000000000000
00000000228210F0: ESID 0000000000000000 VSID 0000000000000000
0000000022821100: ESID 0000000000000000 VSID 000000000010488 V Ks Kp
0000000022821110: ESID 0000000000000000 VSID 0000000000000000
0000000022821120: ESID 0000000000000000 VSID 0000000000000000
0000000022821130: ESID 0000000000000000 VSID 0000000000000000
0000000022821140: ESID 0000000000000000 VSID 0000000000000000
0000000022821150: ESID 0000000000000000 VSID 0000000000000000
(0)> more (^C to quit) ?
<snip>
KDB(0)> ste
Segment Table (STAB)
Select the STAB entry to display by:
1) esid
2) sid
3) dump hash class (input=esid)
4) dump entire stab
Enter your choice: 3
Hash Class to dump (in hex) [esid ok here]: 10
0000000022821800: ESID 0000000000000010 VSID 0000000000000040 V Ks Kp
0000000022821810: ESID 0000000000000000 VSID 0000000000000000
0000000022821820: ESID 0000000000000000 VSID 0000000000000000
0000000022821830: ESID 0000000000000000 VSID 0000000000000000
0000000022821840: ESID 0000000000000000 VSID 0000000000000000
0000000022821850: ESID 0000000000000000 VSID 0000000000000000
0000000022821860: ESID 0000000000000000 VSID 0000000000000000
0000000022821870: ESID 0000000000000000 VSID 0000000000000000
SECONDARY HASH GROUP
0000000022821780: ESID 0000000000000000 VSID 0000000000000000
0000000022821790: ESID 0000000000000000 VSID 0000000000000000
00000000228217A0: ESID 0000000000000000 VSID 0000000000000000
00000000228217B0: ESID 0000000000000000 VSID 0000000000000000
00000000228217C0: ESID 0000000000000000 VSID 0000000000000000
00000000228217D0: ESID 0000000000000000 VSID 0000000000000000
00000000228217E0: ESID 0000000000000000 VSID 0000000000000000
00000000228217F0: ESID 0000000000000000 VSID 0000000000000000
KDB(0)> ste 1
Enter the esid (in hex): 0FFFFFFFF
0000000022821FA0: ESID 0000000000000000 VSID 0000000000000000
KDB(0)>
vmbufst subcommand

Purpose
The vmbufst subcommand displays VMM buf structures.

Syntax
vmbufst [bufaddr]

Parameters
bufaddr Specifies the address of the buf structure to display. If the parameter is omitted, you are prompted to enter it.

The vmbufst subcommand is similar to the general filesystem buf subcommand. It displays a subset of the fields and automatically traverses any buf.av_forw chain.

Aliases
No aliases.

Example
The following is an example of how to use the vmbufst subcommand:

KDB(7)> vmbufst
Enter address of the bufst:34DD79F0 //entered 34DD79F0> vmbufst 34DD79F0
flags........: 000C8001
b_forw........: 00000000  b_back.....: 00000000
av_forw......: 00000000  av_back.....: 00000000
iodone........: 02080A0C  b_wp..........: 00000000
b_dev.........: 000E0003  b_blkno.....: 01882700
b_addr........: 00000000  b_bcount.....: 00001000
b_error......: 00  xmem is at : 00504C78

KDB(7)> buf 34DD79F0 // contrast with the buf cmd
   DEV   VNODE   BLKNO   FLAGS
   0 34DD79F0 000E0003 00000000 01882700 READ SPLIT MPSAFE INITIAL
   forw 00000000 back 00000000 av_forw 00000000 av_back 00000000
   addr 00000000 blkno 01882700
   vp 00000000 flags 000C8001 bcount 00010000 resid 00000000
   work 34E40800 error 00000000 options 00000000 event FFFFFFFF
   iodone: 02080A0C
   start.tv_sec 00015947 start.tv_nsec 00000000
   xmemd.aspace_id FFFFFFFF xmemd.prexflags 00000011
   xmemd.orig_xmem 34DF0030 xmemd.rlist 34DF1030
   orig.aspace_id 00000000 orig.subspace_id 008384CE
   orig.subspace_id2 00000000 orig.uaddr 00000000

KDB(7)>

Another difference between the two commands is that the vmbufst command automatically traverses any av_forw list:

KDB(0)> buf @r5
   DEV   BLKNO   FLAGS
   0 F1000AFD0024F00 8000000000000001 00DE27F0 MPSAFE INITIAL
   forw 0000000000000000 back 0000000000000000
   av_forw F1000AFD002A780 av_back 0000000000000000
   addr 000000000000008000 blkno 000000000DE27F0
   vp 0000000000000000 flags 0000000000000000
bcount 0000000000000020 resid 0000000000000000
work 0000000000000000 error 00000000
options 00000000 event FFFFFFFF
iodone: 034CD180
start.tv_sec 00000000401F4D2B start.tv_nsec 00000000
xmemd.aspace_id 00000000 xmemd.num_sids 00000001
xmemd.subspace_id 000100001914D9000 xmemd.vaddr 0000000000000000
xmemd.prexflags 00000013 xmemd.xp F100000AF0024FB0
xmemd.xp.total 0000000000000020 xmemd.xp.used 0000000000000002
xmemd.xp_s vp 0000000000000000 xmemd.xp.rpn F1000009E25733000
KDB(0)> vmbufst @r5 <also displays the buf at F100000AF002A780>
flags.......: 00000000000C0000
b_forw......: 0000000000000000 b_back....: 0000000000000000
av_forw.....: 0000000000000000 av_back.....: 0000000000000000
iodone......: 000000000034CD180 b_vp........: 0000000000000000
b_dev.......: 0000000000000001 b_blkno.....: 000000000000DE27F0
b_addr......: 0000000000000000 b_bcount....: 0000000000000200
b_error.....: 00 xmem is at : 0000000003016B80

flags.......: 00000000000C0000
b_forw......: 0000000000000000 b_back....: 0000000000000000
av_forw.....: 0000000000000000 av_back.....: 0000000000000000
iodone......: 000000000034CD180 b_vp........: 0000000000000000
b_dev.......: 0000000000000001 b_blkno.....: 000000000000DE2800
b_addr......: 0000000000000000 b_bcount....: 0000000000000200
b_error.....: 00 xmem is aT : 0000000003016B80
KDB(0)>
vmaddr subcommand

Purpose
The vmaddr subcommand displays addresses of VMM structures.

Syntax
vmaddr

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the vmaddr subcommand:
KDB(0)> vmaddr

VMM Addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>H/W PTE</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>H/W PVT</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>H/W PVLIST</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>S/W HAT</td>
<td>A000000000000000</td>
</tr>
<tr>
<td>S/W PFT</td>
<td>4000000000000000</td>
</tr>
<tr>
<td>AHAT</td>
<td>B02A000000000000</td>
</tr>
<tr>
<td>APT</td>
<td>B02C000000000000</td>
</tr>
<tr>
<td>RPHAT</td>
<td>B03C000000000000</td>
</tr>
<tr>
<td>RPT</td>
<td>B03E000000000000</td>
</tr>
<tr>
<td>PDT</td>
<td>B046000000000000</td>
</tr>
<tr>
<td>PFHDATA</td>
<td>B047600000000000</td>
</tr>
<tr>
<td>LOCKANCH</td>
<td>D000000000000000</td>
</tr>
<tr>
<td>SCBs</td>
<td>B047F7C000000000</td>
</tr>
<tr>
<td>ESCBs</td>
<td>B0C7F7C000000000</td>
</tr>
<tr>
<td>LOCKWORDS</td>
<td>B0D8B00000000000</td>
</tr>
<tr>
<td>AMEs</td>
<td>D000000000000000</td>
</tr>
<tr>
<td>LOCK:</td>
<td></td>
</tr>
<tr>
<td>PMAP</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

KDB(0)>
vmdmap subcommand

Purpose
The vmdmap subcommand displays VMM disk maps.

Syntax
vmdmap [slot | Address]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Specifies the Page Device Table (pdt) slot number. This parameter must be a decimal value.</td>
</tr>
<tr>
<td>address</td>
<td>Specifies the address of a specific vmdmap structure to display.</td>
</tr>
</tbody>
</table>

If no parameters are entered, all paging and file system disk maps are displayed. To view a single disk map, enter a slot number.

Aliases
No aliases.

Example
The following is an example of how to use the vmdmap subcommand:

```
KDB(0)> vmdmap
PDT slot [0000] Vmdmap [D0000000] dmsrval [00006003]
mapsize................00020000 freecntr.................0001FF55
agsize..................00000800 agcnt..................00000007
totalags................00000040 lastalloc..........0000004A
maptype.................00000003 clsize.................00000001
clmask................00000100 version.................00000000
btree..................00000000
btree nxt..............00000000
agfree@................D0000030 tree@...............D000000A0
spare1@..............0000002C mpsorsummary@........D0000200
mapsize................00002000 freecntr.................00019980
agsize..................00000800 agcnt..................00000004
totalags................00000004 lastalloc..........00000430
maptype.................00000001 clsize.................00000008
clmask................00000000 version.................00000000
btree..................00000000
btree nxt..............00000000
agfree@................D0000030 tree@...............D000000A0
spare1@..............0000002C mpsorsummary@........D0000200
mapsize................00006000 freecntr.................00004E28
agsize..................00000800 agcnt..................00000008
totalags................0000000C lastalloc..........000000DC
maptype.................00000001 clsize.................00000020
clmask................00000100 version.................00000001
btree..................00000000
btree nxt..............00000000
agfree@................D0000030 tree@...............D000000A0
spare1@..............0000002C mpsorsummary@........D0000200
<snip>
KDB(0)> vmdmap 11
mapsize................00002000 freecntr.................00019980
agsize..................00000800 agcnt..................00000004
totalags................00000004 lastalloc..........0000004A
```
Chapter 24. Display VMM subcommands
vmint subcommand

Purpose
The vmint subcommand displays VMM data for intervals.

Syntax
vmint [ base | list | range ]

Parameters
base | list | range Use one of these optional address input parameters. Identify a base of an interval array, the head of an interval, or the address of a range in an interval to be displayed.

Note: The base and range parameters are typically only used for debugging problems in the vminterval code.

The vmint subcommand displays VMM structure vmintervals information. If no parameter is provided, information on system-wide intervals is displayed.

The vmint subcommand displays one of three types of information when an address input parameter is provided:
- If the address parameter is a base of an interval array, the entire array of vmintervals is displayed.
- If the address parameter is the head of an interval, the vminterval is displayed.
- If the address parameter is the address of one range in an interval, the specific range is displayed.

Aliases
No aliases.

Example
The following is an example of how to use the vmint subcommand:
KDB(0)> vmint

VMM vmint DATA:

VMINT_BADMEM: Memory holes
[270000,10000000]
FFD90000 pages lock @ 010B1420 00000000

VMINT_FIXCOM: Fixed common(BSS) memory
[002937,002C65]
[003A94,003A95]
0000032F pages lock @ 010B13E0 00000000

VMINT_PINOBJ: PINNED object module
[000000,000216]
[000423,000427]
[001000,001333]
[00149C,002C44]
00001CF5 pages lock @ 010B12E0 00000000

VMINT_PAGEDOBJ: PAGED object module
[000288,000410]
[000428,00042B]
[001463,00147E]
[002C65,003A94]
00000FA2 pages lock @ 010B1320 00000000

VMINT_DBGOBJ: DBG object module
[000216,0002BB]
[000427,00042B]
[000485,0005B4]
[001333,001463]
[002C44,002C65]
00000326 pages lock @ 010B1360 00000000

VMINT_INITOBJ: INIT object module
[000410,000423]
00000023 pages lock @ 010B13A0 00000000

220 KDB Kernel debugger and kdb command
Chapter 24. Display VMM subcommands

VMINT_LGPG: Large page memory 00000000 pages lock @ 012EC3C8 00000000
VMINT_FIXLMB: DR non-removeable memory 000190BC pages lock @ 010B16E0 00000000

KDB(0)> vmint 010B1418
FFD90000 pages lock @ 010B1420 00000000
[270000,100000000]

KDB(0)> vmint 010B1688
[270000,100000000] Prev: 010B1418 Next: 010B1438
KDB(0)> vmint 010B1438
[FFFFFFFFFFFFFFFF,FFFFFFFFFFFFFFFF] Prev: 010B1688 Next: 010B1258

KDB(0)> vmint 010B1258
vminterval array based at 010B1258
srad: 0000 freebase: 0
freelist has 80 items starting with 010B1858
freelist lock @ 010B12A0 00000000
0001CF5 pages lock @ 010B12E0 00000000
[000000,000216]
[000423,000427]
[001000,001333]
[00149C,002C44]
00000FA2 pages lock @ 010B1320 00000000
[00028B,000410]
[000428,00042B]
[001463,00147E]
[002C65,003A94]
00000326 pages lock @ 010B1360 00000000
[000216,00028B]
[000427,000428]
[000485,0005B4]
[001333,001463]
[002C44,002C65]
00000023 pages lock @ 010B13A0 00000000
[000410,000423]
[000428,00042D]
[00147E,00148B]
[003A94,003A95]
0000032F pages lock @ 010B13E0 00000000
[002937,002C65]
[003A94,003A95]
FFD90000 pages lock @ 010B1420 00000000
[270000,100000000]
000190BC pages lock @ 010B16E0 00000000
[000000,000A14]
[000C14,000C1B]
[000C48,000C5B]
vmker subcommand

Purpose
The vmker subcommand displays virtual memory kernel data.

Syntax
vmker [-pta] [-dr] [-seg]

Parameters
-pta Displays the Page Table Area (PTA) data.
-dr Displays dynamic memory reconfiguration related data.
-seg Displays VMM segment data.

General VMM kernel data is displayed when no parameter is supplied. All three flags are optional.

Aliases
No aliases.

Example
The following is an example of how to use the vmker subcommand:

KDB(1)> vmker
VMM Kernel Data:
  (use [-pta] | [-dr] [-seg] for specific info)
  rsvd pgsp blks (psrsvdblks) : 00000200
  total page frames (npages) : 00280000
  bad page frames (badpages) : 00000009
  good page frames (goodpages) : 00280000
  ipl page frames (iplpages) : 00280000
  total pgsp blks (numpsblks) : 00020000
  free pgsp blks (psfreeblks) : 0001FE08
  rsvd page frames (pfrsvdblks) : 00080000
  fetch protect (nofetchprot) : 00000000
  max file pageout (maxpout) : 00000000
  min file pageout (minpout) : 00000000
  repage table size (rptsz) : 00010000
  next free in rpt (rptfree) : 00000000
  repage decay rate (rpdecay) : 0000005A
  global repage cnt (sysrepage) : 00000000
  swhashmask (swhashmask) : 01FFFFF
  cachealign (cachealign) : 00010000
  overflows (overflows) : 00000000
  reloads (reloads) : 00000247
  compressed files (noflush) : 00000000
  extended iplclb (iplcbxptr) : 0000000000000000
  alias hash mask (ahashmask) : 0000FFFFF
  max pgs to delete (pd_npagess) : 00001000
  vrlid xlate hits (vrlidhits) : 00000000
  vrlid xlate misses (vrlidmisses) : 00000010
  pgsp bufst waits (psbuwaitcnt): 00000000
  fsys bufst waits (fsbuwaitcnt): 00000084
  rsys bufst waits (rsbuwaitcnt): 00000000
  xpager bufst waits(xpagerbuwaitcnt): 00000000
  phys_mem(s) (phys_mem[0]): 00280000
  phys_mem(s) (phys_mem[1]): FFFFFF
  phys_mem(s) (phys_mem[2]): 00000000
THRPGIO buf wait (_waitcnt) : 00000000
THRPGIO partial cnt (_partialcnt): 00000000
THRPGIO full cnt (_fullcnt) : 00000000
num lgpg's free'd (nlgpgfreed) : 00000000
KDB(1)> vmker -pta

VMM PTA Related Data:

  total pgsp blks (numpsblks) : 00020000
  free pgsp blks (psfreeblks) : 0001FE08
  pta kproc tid (ptakproc_tid) : 0002504B
  # of ptasegments (numptasegs) : 00000001
  ptaseg(s) (ptasegs[1]) : F100000050000000 sid:00020002 sidx:00000002
KDB(1)> vmker -pta

VMM Segment Related Data:

  vmm srval (vmmsrval) : 10001400
  ram disk srval (ramdsrval) : 00000000
  kernel ext srval (kexsrval) : 00000000
  iplcb vsid (iplcbvsmh) : 1F0FF000
  offset of iplcb (iplcboff) : 00000000
  hashbits (hashbits) : 00000015
  hashmask (hashmask) : 001FFFFF
  hash shift amount (stobits) : 00000010
  base config seg (bconfsrval) : 1E0FFE40
  shadow srval (ukernsrval) : 00000000
  kernel srval (kernsrval) : 00000400
  STOI/ITOS mask (stoinmask) : 00000000
  STOI/ITOS sid mask(stoinio) : 00000000
  mallocvmh (mallocvmh) : 1B013B400
  # of ptasegments (numptasegs) : 00000001
  ptaseg(s) (ptasegs[1]) : F100000050000000
KDB(1)> vmker -pta

VMM DR Related Data:

  total page frames (nrpages) : 00280000
  bad page frames (badpages) : 00000009
  good page frames (goodpages) : 00280000
  ipl page frames (iplpages) : 00280000
  rsvd page frames (pfrsvdblks) : 00080000
  DR mem adds (addlmbs) : 00000000
  DR mem removes (rmlmbs) : 00000000
  DR fixlmb migrates (fixlmb) : 00000000
  DR reloaeds ena (ena_rldmigmiss): 00000000
  DR reloaeds dis (dis_rldmigmiss): 00000000
  DR refcntmiss (migrefcntmiss): 00000000
  DR migr trans (migtransients) : 00000000
  DR mark trans (marktransients) : 00000000
  DR migr misses (vlookmigmiss) : 00000000
  DR vmm migrates (vmm_migrates) : 00000000
  DR serv migrates(serv_migrates): 00000000
  DR vmppool adds (add_vmps) : 00000000
  DR vmppool removes (rem_vmps) : 00000000
  DR vmppool dormant (dor_vmps) : 00000000
(1)> more (^C to quit) ?

KDB Kernel debugger and kdb command
vmlocks subcommand

Purpose
The `vmlocks` subcommand displays VMM spin lock data.

Syntax
`vmlocks`

Parameters
No parameters.

Aliases
`vmlock`, `vl`

Example
The following is an example of how to use the `vl` alias for the `vmlocks` subcommand:

```
KDB(0) > vl
GLOBAL LOCKS

pmap   lock at 00000000 FREE
vmap   lock at 0B076100 FREE
ame   lock at 0B076180 FREE
rpt global lock at 0B076200 FREE
rpt pool lock [0] @ 0B076280 FREE
rpt pool lock [1] @ 0B076284 FREE
rpt pool lock [2] @ 0B076288 FREE
rpt pool lock [3] @ 0B07628C FREE
rpt pool lock [4] @ 0B076290 FREE
rpt pool lock [5] @ 0B076294 FREE
rpt pool lock [6] @ 0B076298 FREE
rpt pool lock [7] @ 0B07629C FREE
rpt pool lock [8] @ 0B0762A0 FREE
rpt pool lock [9] @ 0B0762A4 FREE
rpt pool lock [10] @ 0B0762AB FREE
rpt pool lock [11] @ 0B0762AC FREE
rpt pool lock [12] @ 0B0762B0 FREE
rpt pool lock [13] @ 0B0762B4 FREE
rpt pool lock [14] @ 0B0762B8 FREE
rpt pool lock [15] @ 0B0762BC FREE
alloc   lock at 0B076300 FREE
apt    lock at 0B076380 FREE
pdt alloc lock at 0B076400 FREE
pdt iolist lock at 0B076480 FREE
comp   lock at 0B076500 FREE
zq     lock at 006F09C8 FREE
lw     lock at 006F0BE0 FREE

MEMORY POOLS & FRAMESET LOCKS

VMPOOL 00

mempool[00000000]: LRU    lock at 01FAA004 FREE
frameset[00000000]: free nfr lock @ 01F94000 FREE
frameset[00000001]: free nfr lock @ 01F94080 FREE

SCOREBOARD

scoreboard cpu 0 :
hint.....................00000000
```
00: empty
01: empty
(0)> more (^C to quit)
02: empty
03: empty
04: empty
05: empty
06: empty
07: empty
scoreboard cpu 1:
hint................................00000000
00: empty
01: empty
02: empty
03: empty
04: empty
05: empty
06: empty
07: empty
KDB(0)>
vmlog subcommand

Purpose
The vmlog subcommand displays the current VMM error log entry.

Syntax
vmlog

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the vmlog subcommand:

```
KDB(0)> vmlog //display VMM error log entry
Most recent VMM errorlog entry
Error id = DSI_PROC
Exception DSISR/ISISR = 40000000
Exception srval = 007FFFFFFF
Exception virt addr = FFFFFFFF
Exception value = 0000000E
KDB(0)> dr iar //display current instruction
iar : 01913DF0
 01913DF0 lwz r0,0(r3)      r0=00001030,0(r3)=FFFFFFF
KDB(0)>```

Chapter 24. Display VMM subcommands 227
vmpool subcommand

Purpose
The vmpool subcommand displays VMM information for resource pools.

Syntax
vmpool {[-l | -d | -f] * | vmpool_id}

Parameters
- I Indicates that the SYSVMP_LGPG type anchor should be accessed.
- d Indicates that the SYSVMP_DORM type anchor should be accessed.
- f Indicates that the SYSVMP_FREE type anchor should be accessed.
  * Indicates that the summary information is to be displayed.
  vmpool_id Indicates the specific vmpool identifier.

The vmpool subcommand displays VMM data for resource pools (struct vmpool_t). Use the asterisk (*) parameter to display summary information. The information you select to display can be modified by including one of the flags. If none of the flags are used, the SYSVMP_NORMAL-type anchor is accessed.

You can also use the vmpool subcommand to display information for a specific vmpool identifier.

Aliases
No aliases.

Example
The following is an example of how to use the vmpool subcommand:

KDB(1)> vmpool *
VMM Resource Pools Data:
VMP NEXT LRPAGES MEMPOOLS FPMP MEMP_VMINT
00 -1 000026549F 001: 000 002 F1000142000000

KDB(1)> vmpool -l *
No vmpools on this list.

KDB(1)> vmpool -f *
VMM Resource Pools Data:
VMP NEXT
01 02
02 03
03 04
04 05
05 06
06 07
07 08
08 09
09 0A
0A 0B
0B 0C
0C 0D
0D 0E
0E 0F
0F -1

KDB(1)> vmpool 2
VMPOOL 02 (addr = 000000000027C9B0)

- number of LRUable pages (npages_lru): 00000000
- sradid (srad_id): 00000000
- first memory pool (memp_first): FFFFFFFF
- number of memory pools (nb_mempool): 00000000
- number of frame sets / memp (nb_frs_per_memp): 00000000
- first nfr on lgpg freelist (lgpg_free): 0000000000000001
- number of frames on lgpg freelist (lgpg_numfrb): 0000000000000000
- total # of lgpg frames (npages_lg): 0000000000000000
- addr of vmintervals array (vmint): 0000000000000000
- addr of freemem list (freemem): 0000000000000000
- addr of usedmem list (usedmem): 0000000000000000
- affinity_list (affinity_list): 000000000027C9F0
  - NULL
- next vmpool (next): 03
- next lgpg vmpool (next_lgpg): 00
- last_[memp/frs]_ecpus: 0000 / 0000
- vmpool flags (flags): 00000000
- large page frb lock @ 000000000027CA50 00000000
- memp frs dr lock @ 000000000027CA58 00000000

KDB(1)>
vmstat subcommand

Purpose
The vmstat subcommand displays virtual memory statistics.

Syntax
vmstat

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the vmstat subcommand:
KDB(0)> vmstat

VMM Statistics:

page faults (pgexct) : 00105695
page reclaims (pgrclm) : 00000000
lockmisses (lockexct) : 00000000
backtracks (backtrks) : 0000C2A2
pages paged in (pageins) : 00004824
pages paged out (pageouts) : 0000CEFA
paging space page ins (pgspgins) : 00000000
paging space page outs (pgspgouts) : 00000000
start I/Os (numsios) : 0000E251
iodones (numiodone) : 0000CFAA
zero filled pages (zerofills) : 0007764B
executable filled pages (exfills) : 00000000
pages examined by clock (scans) : 00000000
clock hand cycles (cycles) : 00000000
page steals (pgsteals) : 00000000
free frame waits (freewts) : 00000000
extend XPT waits (extendwts) : 00000000
pending I/O waits (pendiowts) : 000028C7

VMM Statistics:

total virtual pgs (numvpages): 000000000000BA03
pages in use for wseg (numwseguse): 000000000000881F
pages in use for pseg (numpseguse): 0000000000002D1F
pages in use for clseg (numclseguse): 0000000000007D9F
pages pinned for wseg (numwsegpin): 00000000000037D8
pages pinned for pseg (numpsegpin): 0000000000000000
pages pinned for clseg (numclsegpin): 0000000000000000
ping-pons: source => alias (pings) : 00000000
ping-pons: alias => source (pongs) : 00000000
ping-pons: alias => alias (pangs) : 00000000
ping-pons: alias page del (dpongs) : 00000000
ping-pons: alias page write (wpons) : 00000000
ping-pong cache flushes (cachef) : 00000000
ping-pong cache invalidates (cachei) : 00000000
hardware large page size (lgpg_size): 00000000
total num of large pages (lgpg_cnt): 0000000000000000
num free large pages (lgpg_numfrb): 0000000000000000
large page high water cnt (lgpg_hi): 0000000000000000
large page in-use cnt (lgpg_inuse): 0000000000000000

230 KDB Kernel debugger and kdb command
num reserved sids  (numspecseg) : 00000000000000000
num free reserved sids (numspecfree): 00000000000000000
reserved sids hi-water  (specsegshi): 00000000000000000
success mem guards (memgrd_suc_pgs): 00000000000000000
failed mem guards  (memgrd fail_pgs): 00000000000000000
KDB(0)>

VMM Memory Limits:

Total available memory (4K frames) : 000800000
4K number of frames : 000799F9
4K frames pinned : 00005F6F
4K system pinnable frames remaining: 00058559
4K user pinnable frames remaining : 00058E6E

Free paging space (in 4K blocks) : 0001FDA6
Paging space SIGDANGER level : 00001000
Paging space SIGKILL level : 00000400
vmthrpgio subcommand

Purpose
The vmthrpgio subcommand provides VMM support of thread/base level page I/O commands.

Syntax
vmthrpgio

Parameters
No parameters.

When you enter the vmthrpgio subcommand, the following options are displayed:
1) display a given thrpio frame structure (user provides the address)
2) display the ut_pgio_fields of the current thread
3) display THRPIO bufstructs. The user provides the address of
   a struct bufthrio. Any av_forw chain is traversed, displaying
   each struct bufthrio.

Aliases
No aliases.

Example
No example.
vmwait subcommand

Purpose
The vmwait subcommand displays VMM wait status.

Syntax
vmwait [effectiveaddress]

Parameters
effectiveaddress Specifies the effective address for a wait channel. Symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is used, you are prompted for the wait address.

Aliases
No aliases.

Example
The following is an example of how to use the vmwait subcommand:

KDB(0)> th -w WPGIN
  SLOT  NAME  STATE  TID  PRI  RQ  CPUID  CL  WCHAN
pvthread+004600 140 sync  SLEEP 008CF1 03C 1 0 0 B048CCA0
KDB(0)> vmwait B048CCA0
VMM Wait Info
Waiting on persistent segment I/O level (v_iowait), sidx = 000003CB
KDB(0)>

Chapter 24. Display VMM subcommands 233
vrld subcommand

Purpose
The vrld subcommand displays the VMM reload translate table. This information is used only on the SMP POWER-based machine to prevent VMM reload dead-lock.

Syntax
vrld

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the vrld subcommand:

KDB(0)> vrld

freepno: 0A, initobj: 0008DA58, *initobj: FFFFFFFF

[00] sid: 00000000, anch: 00
  (00) spno:00000000, epno:00000097, nfr:00000000, next:01
  (01) spno:00000098, epno:000000A8, nfr:00000098, next:02
  (02) spno:FFFFFFF, epno:000001F6, nfr:000001DD, next:03
  (03) spno:000001F7, epno:000001FA, nfr:000001F7, next:04
  (04) spno:0000038C, epno:000003E3, nfr:00000323, next:FF

[01] sid: 00000041, anch: 06
  (06) spno:00003400, epno:0000341F, nfr:000006EF, next:05
  (05) spno:00003800, epno:00003A8F, nfr:000003F0, next:08
  (08) spno:00006800, epno:00006800, nfr:0000037C, next:07
  (07) spno:00006820, epno:00006820, nfr:0000037B, next:09
  (09) spno:000069C0, epno:000069CC, nfr:0000072F, next:FF

[02] sid: FFFFFFFF, anch: FF

[03] sid: FFFFFFFF, anch: FF

KDB(0)>
vsidd subcommand

Purpose
The vsidd subcommand displays memory using a vsid (virtual segment identifier) and byte offset addressing format.

Syntax
vsidd {vsid:offset} [count] [,w,d]

Parameters
- **vsid:offset**: Identifies the memory location to be displayed. The vsid parameter indicates which segment to access, and the offset is the number of bytes into the segment from which to begin displaying. These parameters are required.
- **count**: Indicates the number of display units (4-byte words or 16-byte double words) to display. If count is omitted, one line (32-bytes) of data is displayed.
- **,w**: Indicates that the display unit is 4-byte words.
- **,d**: Indicates that the display unit is 8-byte double words.

Note: For the 32-bit kernel, the default display unit is 4 bytes. For the 64-bit kernel, the default display unit is 8 bytes. In both cases, the page must be in memory.

Aliases
sidd

Example
The following is an example of how to use the vsidd subcommand:

Display starting at offset 0x80 from the segment containing the IPL control block (example vsid of 1F0FFF) on the 64-bit kernel:

KDB(0)> vsidd 1F0FFF:80 8
001F0FFF:00000000 524F5349504C200A 0000000000131F0 ROSIPL ......1.
001F0FFF:00000090 0000010C00000007 0000032800000598 ..........(....
001F0FFF:000000A0 0000000000000000 0000000000000000 ............
001F0FFF:000000B0 0000000000000000 0000000000000000 ............
KDB(0)> vsidd 1F0FFF:80 8,w
001F0FFF:00000000 524F5349 504C200A 00000000 000131F0 ROSIPL ......1.
001F0FFF:00000090 00000001 00000007 00000328 00000598 ..........(....
KDB(0)>
vsidm subcommand

Purpose
The vsidm subcommand modifies memory using a vsid (virtual segment identifier) and the byte offset addressing format.

Syntax
vsidm {vsid:offset} [,w,d]

Parameters

vsid:offset  Identifies the memory location to be modified. The vsid parameter indicates which segment to access, and the offset is the first byte to access. These parameters are required.

,w  Indicates that the modification unit is 4-byte words.

,d  Indicates that the modification unit is 8-byte double words.

Note: For the 32-bit kernel, the default modification unit is 4 bytes. For the 64-bit kernel, the default modification unit is 8 bytes. In both cases, the page must be in memory.

This vsidm subcommand works like other memory-modification commands. The current word (or double word) at the target location is displayed. If you enter a new value, the memory location is changed. If you press Enter without typing a value, the value in the memory location remains unchanged and the next location is displayed for modification. When you type a period (.), the command terminates.

Aliases

sidm

Example
The following is an example of how to use the vsidm subcommand:

Modify starting at offset 0x80 from the segment containing the IPL control block (example vsid of 1F0FFF) on the 64-bit kernel, using word (4 byte) units

KDB(0)> vsidm 1F0FFF:80,w
001F0FFF:00000080:  524F5349  4B444249
001F0FFF:00000084:  504C200A  <press enter>
001F0FFF:00000088:  00000000  = .
KDB(0)> vsidm 1F0FFF:80,w
001F0FFF:00000080:  4B444249  504C200A  00000000  000131F0  KDBIPL .......1.
KDB(0)> vsidm 1F0FFF:80
001F0FFF:00000080:  4B444249  504C200A  = 524F5349504C200A
001F0FFF:00000088:  0000000000131F0  = .
KDB(0)> vsidm 1F0FFF:80,w
001F0FFF:00000080:  524F5349  504C200A  00000000  000131F0  ROSIPL .......1.
KDB(0)>
### zproc subcommand

**Purpose**
The `zproc` subcommand displays information about the VMM zeroing kproc.

**Syntax**
```
zproc
```

**Parameters**
No parameters

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `zproc` subcommand:

```
KDB(1)> zproc  //display VMM zeroing kproc
VMM zkproc pid = 63CA  tid = 63FB
Current queue info
  Queue resides at 0x0009E3E8 with 10 elements
  Requests 16800 processed 16800 failed 0
  Elements
      sid  pno  npg  pno  npg
    0 -FF000000 FF000000 FF000000 FF000000 FF000000
    1 -FF000000 FF000000 FF000000 FF000000 FF000000
    2 -FF000000 FF000000 FF000000 FF000000 FF000000
    3 -FF000000 FF000000 FF000000 FF000000 FF000000
    4 -FF000000 FF000000 FF000000 FF000000 FF000000
    5 -FF000000 FF000000 FF000000 FF000000 FF000000
    6 -FF000000 FF000000 FF000000 FF000000 FF000000
    7 -FF000000 FF000000 FF000000 FF000000 FF000000
    8 -FF000000 FF000000 FF000000 FF000000 FF000000
    9 -FF000000 FF000000 FF000000 FF000000 FF000000
```
**drlist subcommand**

**Purpose**
The `drlist` subcommand displays VMM data for a `drlist_t` structure.

**Note:** This command is not available on the uni-processor (unix_up) kernel.

**Syntax**
```
drlist [address]
```

**Parameters**
- `address` Specifies the memory location to be displayed as a `drlist_t` structure.

The `drlist` command is used to display a `drlist_t` structure. If no parameter is given, the global kernel anchor is examined and the `drlist_t`, (if any), is displayed. If there is no valid outstanding DRlist, a message is displayed.

If the `address` parameter is given, the memory location is displayed as a `drlist_t` structure.

**Aliases**
```
drl
```

**Example**
The following is an example of how to use the `drlist` subcommand:
```
KDB(0)> drlist
DRlist 0 F10010F0164A0700

start frame....... 0000000000270000
end frame......... 0000000000280000
swpfts........... F20080001EA0000
swpfte........... F20080001F00000
pyts............. F20080002138000
pyte.............. F20080002140000
pftpages......... 0000000000000000
vmpool_id........ 00000000
memop............. 00000000
flags............. 00000000
freefwd........... 0000000000000000
freebwd........... 0000000000000000
nfree............. 0000000000000000
lruuptr........... 0000000003A61430
lruvisits......... 0000000000000000
maxvisits......... 0000000000000000
lrusteals......... 0000000000000000
maxpouts......... 0000000000000000
lruouts........... 0000000000000000
lrupass........... 00000000
addnfr........... 0000000000000000
lock............... 0000000000000000

KDB(0)> 238
```
lrustate subcommand

Purpose
The lrustate displays the lru daemon control variables.

Note: These variables reside on the respective lru daemon stack, and only have valid values while the lru
daemon is actively running.

Syntax
lrustate [ mempool id ]

Parameters
mempool id Is the memory pool identifier that corresponds to the lru daemon whose state you want to
examine.

Aliases
lru

Example
The following is an example of how to use the lru alias for the lrustate subcommand:
KDB(0)> lru -?
lru <mempool id>
KDB(0)> lru 0
LRU State @00B1F520 for mempool 0
*** this is on the MST stack & only valid if fblru running ***

LRU Start nfr (lru_start) : 00000000
mempools first nfr (lru_firstnfr) : 00000000
numfrb this mempool (lru_numfrb) : 00000004
number of steals (lru_steals) : 00000000
page goal to steal (lru_goal) : 0000001B
npages scanned (lru_nbscan) : 00000002
LFBLRU or CFBLRU (lru_type) : 00000000 LFBLRU
scans of start nfr (lru_scan_start_cnt) : 00000000
1ru revolutions (lru_rev) : 00000000
last buckt<bucketsz(lru_small_mem_wrap) : 00000000
fileonly mode (lru_fileonly) : 00000000
progress guaranteed (lru_progress) : 00000001
fault color (lru_fault_col) : 00000173, 371
steal color (lru_steal_col) : 00000173, 371
nbuckets scanned (lru_nbuckets) : 00000001
1ru mode (lru_mode) : 00000000
wlm regul enabled? (lru_wlm_is_enabled) : 00000001 WLM Regul is ON
request type (lru_rq) : 00000009
drbit before pgout (lru_drbit) : 00000000
1ru_dr running (lru_dr) : 00000000
start ccb (lru_start_ccb) : 00000000, 0
ccb passl left off (lru_pl_ccb) : 00000000, 0
current ccb (lru_cur_ccb) : 00000000, 0
KDB(0)>
Chapter 25. Address translation subcommands

The subcommands in this category can be used to display address translation information, display and modify ibat and dbat registers on POWER-based machines, and display and modify Segment Lookaside Buffer (SLB) information. These subcommands include the following:

- `vt`
- `tv`
- `slb`
- `mslb`
- `dbat`
- `ibat`
- `mdbat`
- `mibat`
tr and tv subcommands

Purpose
The tr and tv subcommands display address translation information. The tr subcommand provides a short format and the tv subcommand provides a detailed format.

Syntax
tr effectiveaddress

Parameters
effectiveaddress Specifies the effective address for which translation details are to be displayed. Use symbols, hexadecimal values or hexadecimal expressions to specify the address.

For the tv subcommand, all double-hashed entries are dumped when the entry matches the specified effective address. Corresponding physical address and protections are displayed. Page protection (the K bit and the PP bits) is displayed according to the current segment and machine state register values.

Aliases
No aliases.

Example
The following is an example of how to use the tr and the tv subcommands:

KDB(0)> nm pvthread
Symbol Address : F1000588D0000000
TOC Address : 01505F20
KDB(0)> tr pvthread
Physical Address = 000000007F964000
KDB(0)> tv pvthread
starting
kd_get_vsid 1F88D
eaddr F1000588D0000000 sid 00000000001F88D vpage 0000000000000000 hashl 0001F88D
p64pte_cur_addr 0000000002FC4680 sid 00000000001F88D avp 00 hsel 0 valid 1
rpn 0000000000000000 refbit 1 modbit 1 wimg 2 key 0
____ 000000007F964000 ____ K = 0 PP = 00 ==> read/write

eaddr F1000588D0000000 sid 00000000001F88D vpage 0000000000000000 hash2 00020772
Physical Address = 000000007F964000
KDB(0)>
slb subcommand

Purpose
The slb subcommand displays Segment Lookaside Buffer (SLB) information.

Syntax
slb [-r] [entry]

Parameters
- -r  Specifies that the current register contents of the SLBs should be displayed. If there are any SLB values, the slb subcommand usually displays them for the current context, but does not display the contents of the registers.
  Note: This flag is only supported for the KDB kernel debugger.
entry  Specifies the SLB entry to display. If this parameter is not used, all of the SLBs are displayed.

If the underlying hardware platform does not support SLBs, the slb subcommand displays a message indicating that the subcommand is unavailable.

Aliases
No aliases.

Example
The following is an example of how to use the slb subcommand:

KDB(0)> slb
00 0000000008000000 0000000000000040 V 01 F000000028000000 000000021002000 V
02 F000000039000000 000000011E0040 I 03 FFFFFFFFFF03000000 000001C10F0C80 V
04 FFFFFFFF000000001A04080 I 05 FFFFFFFFFF02000000 000000112D0D80 I
06 FFFFFFFF0000000011113108 I 07 0000000000000000 0000000000000000 I
08 0000000000000000 0000000000000000 I 09 0000000000000000 0000000000000000 I
0A 0000000000000000 0000000000000000 I 0B 0000000000000000 0000000000000000 I
0C F100000918000000 000000001E090E40 V 0D F10000878000000 000000015878540 V
0E F10000878000000 000000015878540 V 0F F1000089C800000 000000019089940 V
10 F1000000EB000000 0000000000000000 I 11 F1000000EB000000 00000001800B840 V
12 F1000000EB000000 0000000000000000 I 13 F1000000EB000000 00000002000B240 V
14 F1000000EB000000 0000000000000000 I 15 F1000000EB000000 00000001800F840 V
16 F1000000EB000000 0000000000000000 I 17 F1000000EB000000 00000001A0FA40 V
18 F20001001B000000 0000000160013108 V 19 F20001001B000000 00000000200420 V
1A F20001001B000000 0000000160013108 V 1B F1000000EB000000 00000001800840 V
1C F1000000100000000 00000001900A4000 I 1D F1000000100000000 00000002000240 V
1E F1000000100000000 00000001900A4000 I 1F F1000000100000000 00000002200240 V
20 F20001001B000000 0000000160013108 V 21 F20001001B000000 00000001800840 V
22 F20001001B000000 0000000160013108 V 23 F1000000100000000 00000002000240 V
24 F1000000100000000 00000001900A4000 I 25 F1000000100000000 00000002200240 V
26 F1000000100000000 00000001900A4000 I 27 F1000000100000000 00000002200240 V
28 F1000000100000000 00000001900A4000 I 29 F1000000100000000 00000002200240 V
2A F1000000100000000 00000001900A4000 I 2B F1000000100000000 00000002200240 V
2C F1000000100000000 00000001900A4000 I 2D F1000000100000000 00000002200240 V
2E F1000000100000000 00000001900A4000 I 2F F1000000100000000 00000002200240 V
30 F1000000100000000 00000001900A4000 I 31 F1000000100000000 00000002200240 V
32 F1000000100000000 00000001900A4000 I 33 F1000000100000000 00000002200240 V
34 F1000000100000000 00000001900A4000 I 35 F1000000100000000 00000002200240 V
36 F1000000100000000 00000001900A4000 I 37 F1000000100000000 00000002200240 V
38 F1000000100000000 00000001900A4000 I 39 F1000000100000000 00000002200240 V
3A F1000000100000000 00000001900A4000 I 3B F1000000100000000 00000002200240 V
3C F1000000100000000 00000001900A4000 I 3D F1000000100000000 00000002200240 V
3E F1000000100000000 00000001900A4000 I 3F F1000000100000000 00000002200240 V
KDB(0) > slb 3
03 FFFFFFFF08000000 00000001C109C080 V
> valid
esid = 00000000FFFFFFF0
vsid = 00000000001C109C
KsKp = 00 NLC = 001
KDB(0) >
mslb subcommand

Purpose
The **mslb** subcommand modifies (Segment Lookaside Buffer) SLB information.

Syntax
`mslb [-r] [entry]`

Parameters
- **-r** 
  Specifies that the current register contents of the SLB should be modified. If the **-r** flag is not used, the **mslb** subcommand changes the SLB value for the current context.

  **Note:** The **-r** flag is only supported for the KDB kernel debugger.

- **entry** 
  Indicates the specific SLB entry to modify. This value is a decimal value. If no **entry** parameter is provided, the subcommand defaults to entry number 0.

The update procedure is identical to other modification subcommands. The current value is displayed, and:

- The value can be altered.
- The value can be left unmodified if you press Enter. Pressing Enter causes the next SLB to be displayed. The next SLB is displayed only if no **entry** parameter is entered. If you modify a specific SLB entry, the subcommand terminates after it advances past the virtual segment identifier (VSID) double word.
- The **mslb** subcommand can be terminated if you enter a period (.)

The SLB is treated as two 8-byte double words, referred to as the effective segment identifier (ESID) and the virtual segment identifier (VSID) respectively. If the underlying hardware platform does not support SLBs, the **mslb** subcommand displays a message indicating that the subcommand is unavailable.

Aliases
No aliases.

Example
The following is an example of how to use the **mslb** subcommand:

```
KDB(1)> slb 3
03 0000000000000000 000000FFFFFFF000 I
esid = 0000000000000000
vsid = 00000000FFFFFFF
KsKp = 00 NLC = 000
KDB(1)> mslb 3
03 0000000000000000 000000FFFFFFF000 I Entry ESID = FFFFFFFF08000000 <entered new value FFFFFFFF08000000>
03 FFFFFFFF08000000 000000FFFFFFF000 V Entry VSID = 000000001C109C080 <entered new value 000000001C109C080>
KDB(1)> slb 3
03 FFFFFFFF08000000 00000001C109C080 V
> valid
esid = 00000000FFFFFFF0
vsid = 00000000001C109C
KsKp = 00 NLC = 001
KDB(1)>
```
dbat subcommand

Purpose
On POWER-based machines that implement the block address translation facility, the dbat subcommand displays dbat registers.

Syntax
dbat [index]

Parameters
index Specifies the dbat register to display. Valid values are 0 through 3. If no parameter is specified all dbat registers are displayed.

Aliases
No aliases.

Example
The following is an example of how to use the dbat subcommand:

KDB(0)> dbat
DBAT0 0000000040001FFE 00000000C000003A
    bepi 000000002000 brpn 000000006000 b1 07FF vs 1 vp 0 wimg 7 pp 2
eaddr = 0000000040000000, paddr = 00000000C0000000 size = 262144 KBytes [Supervisor state]
DBAT1 0000000050001FFE 00000000C000003A
    bepi 000000002800 brpn 000000006000 b1 07FF vs 1 vp 0 wimg 7 pp 2
eaddr = 0000000050000000, paddr = 00000000C0000000 size = 262144 KBytes [Supervisor state]
DBAT2 0000000000000000 0000000000000000
    bepi 000000000000 brpn 000000000000 b1 0000 vs 0 vp 0 wimg 0 pp 2
DBAT3 0000000000000000 0000000000000000
    bepi 000000000000 brpn 000000000000 b1 0000 vs 0 vp 0 wimg 0 pp 0
KDB(0)> dbat 0
DBAT0 0000000040001FFE 00000000C000003A
    bepi 000000002000 brpn 000000006000 b1 07FF vs 1 vp 0 wimg 7 pp 2
eaddr = 0000000040000000, paddr = 00000000C0000000 size = 262144 KBytes [Supervisor state]
**ibat subcommand**

**Purpose**
On POWER-based machines that implement the block address translation facility, the **ibat** subcommand can be used to display ibat registers.

**Syntax**
ibat [index]

**Parameters**

index Specifies the ibat register to display. Valid values are 0 through 3. If no parameter is specified, all ibat registers are displayed.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the **ibat** subcommand:

KDB(0)> ibat 0
IBAT0 0000000000000000 0000000000000000
bepl 0000000000000000 brpn 0000000000000000 bl 0000 vs 0 vp 0 wimg 0 pp 0
KDB(0)>
mdbat subcommand

Purpose
The mdbat subcommand is used to modify the dbat register. The processor data bat register is modified immediately. The word containing the valid bit is set last.

Syntax
mdbat [index]

Parameters
index Specifies the dbat register to modify. Valid values are 0 through 3.

If no parameter is entered, you are prompted for the values for all dbat registers. If a parameter is specified for the mdbat subcommand, you are only prompted for the new values for the specified dbat register.

You can input both the upper and lower values for each dbat register or you can press Enter for these values. If the upper and lower values for the register are not entered, the user is prompted for the values for the individual fields of the dbat register. To stop entering values, you type a period (.) and press Enter at any prompt.

Aliases
No aliases.

Example
The following is an example of how to use the mdbat subcommand on a PowerPC 604 RISC Microprocessor:

    KDB(0)> mdbat 2 //alter bat register 2
    BAT register, enter <RC> twice to select BAT field, enter <*> to quit
    DBAT2 upper 00000000 =
    DBAT2 lower 00000000 =
    BAT field, enter <RC> to select field, enter <*> to quit
    DBAT2.bepi: 00000000 = 00007FE0
    DBAT2.brpn: 00000000 = 00007FE0
    DBAT2.bl : 00000000 = 0000001F
    DBAT2.vs : 00000000 = 00000001
    DBAT2.vp : 00000000 = <CR/LF>
    DBAT2.wimg: 00000000 = 00000003
    DBAT2.pp : 00000000 = 00000002
    DBAT2 FFC0007E FFC0001A
    bepi 7FE0 brpn 7FE0 bl 001F vs 1 vp 0 wimg 3 pp 2
eaddr = FFC00000, paddr = FFC00000 size = 4096 KBytes [Supervisor state]
    KDB(0)> mdbat 2 //clear bat register 2
    BAT register, enter <RC> twice to select BAT field, enter <*> to quit
    DBAT2 upper FFC0007E = 0
    DBAT2 lower FFC0001A = 0
    DBAT2 00000000 00000000
    bepi 0000 brpn 0000 bl 0000 vs 0 vp 0 wimg 0 pp 0
**mibat subcommand**

**Purpose**
The `mibat` subcommand is used to modify the ibat register. The processor instruction bat register is changed immediately.

**Syntax**
```
mibat [index]
```

**Parameters**

`index` Specifies the ibat register to modify. Valid values are 0 through 3.

If no parameter is specified, you are prompted for the values for all ibat registers. If a parameter is specified for the `mibat` subcommand, you are only prompted for the new values for the specified ibat register.

Input both the upper and lower values for each ibat register or press Enter to use these values. If the upper and lower values for the register are not entered, you are prompted for the values for the individual fields of the ibat register. You can stop entering values by typing a period (.) at any prompt and pressing Enter.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `mibat` subcommand on a PowerPC 604 RISC Microprocessor:

```
KDB(0)> mibat 2
BAT register, enter <RC> twice to select BAT field, enter <.> to quit
IBAT2 upper 00000000 = <CR/LF>
IBAT2 lower 00000000 = <CR/LF>
BAT field, enter <RC> to select field, enter <.> to quit
IBAT2.bepi: 00000000 = <CR/LF>
IBAT2.brpn: 00000000 = <CR/LF>
IBAT2.bl : 00000000 = 3ff
IBAT2.vs : 00000000 = 1
IBAT2.vp : 00000000 = <CR/LF>
IBAT2.wimg: 00000000 = 2
IBAT2.pp : 00000000 = 2
IBAT 00000FFE 00000012
bepi 0000 brpn 0000 bl 03FF vs 1 vp 0 wimg 2 pp 2
```

```eaddr = 00000000, paddr = 00000000 size = 131072 KBytes [Supervisor state]```
Chapter 26. Loader subcommands

The subcommands in this category display the kernel loader entries, add symbols from loaded kernel extensions to the KDB kernel debugger's symbol name cache, and display or remove symbol tables. These subcommands include the following:

- lke
- stbl
- rmst
- lle
- exp
ike, stbl, and rmst subcommand

Purpose
The **ike** subcommand displays the kernel loader entries and adds symbols from loaded kernel extensions to the symbol name cache that is used for debugging. The **stbl** subcommand displays the symbol tables. The **rmst** subcommand removes a symbol table.

Syntax
ike [-l] [-l32] [-l64] [-p pslot] [-n name] [(-s) {entry | effectiveaddress}] [-a ldr_address]
stbl [sym_slot | ldr_address]
rmst [sym_slot | ldr_address]

Parameters
- **-l** Lists the current entries in the name list cache.
- **-l32** Displays loader entries for 32-bit shared libraries.
- **-l64** Displays loader entries for 64-bit shared libraries.
- **-p pslot** Displays the shared-library loader entries for the process slot indicated. The value for *pslot* must be a decimal process slot number.
- **-n name** Displays the loader entry specified by *name*.
- **-s** Does not display symbols when populating the cache.
- **entry** Specifies a loader entry. The *entry* parameter must be a decimal value. The specified entry is displayed, and the name list cache is loaded with data for that entry.
- **effectiveaddress** Specifies an effective address in the text or data area for a loader entry. The specified entry is displayed and the name list cache is loaded with data for that entry. This address can be a hexadecimal value, a symbol, or a hexadecimal expression.
- **-a ldr_address** Displays the loader entry at the specified address, and loads the name list cache with data for that entry. This address can be a hexadecimal value, a symbol, or a hexadecimal expression.
- **sym_slot** Specifies the slot number. This value must be a decimal number.
- **ldr_address** Specifies the address of a loader entry. The address can be a hexadecimal value, a symbol, or a hexadecimal expression.

During boot phase, KDB kernel debugger is called to load extension symbol tables. When KDB kernel debugger is called, a message is displayed.

The symbol tables that are available to KDB kernel debugger can be listed with the **stbl** subcommand. If this subcommand is invoked without parameters, a summary of all symbol tables is displayed. Details about a particular symbol table can be obtained by supplying a slot number or the effective address of the loader entry to the **stbl** subcommand.

A symbol table can be removed from KDB kernel debugger using the **rmst** subcommand. This subcommand requires that either a slot number or the effective address for the loader entry of the symbol table be specified.

A symbol name cache is managed inside KDB kernel debugger. The cache is filled with function names with the **ike** [-s] { *entry* | *address* } subcommand and the **ike** -a ldr_address subcommand. When this cache is full, old entries are replaced by new entries.

If the **ike** subcommand is invoked without parameters, a summary of the kernel loader entries is displayed. The **ike** subcommand parameters **-l32** and **-l64** can be used to list the loader entries for 32-bit and 64-bit shared libraries, respectively. Details can be viewed for individual loader entries by specifying the following:
### Entry number
- Address of the loader entry with the -a flag
- Address within the text or data area for a loader entry

The name lists contained in the name list cache area can be reviewed by using the -l option.

### Aliases
No aliases.

### Example
The following is an example of how to use the `stbl`, `rmst` and `lke` subcommand when `/unix` and one driver have symbol tables:

**Note:** If the kernel extension is stripped, the symbol table is not loaded in memory.

```
...//during boot phase
  no symbol [/etc/drivers/mdttu_load]
  no symbol [/etc/drivers/fd]
  Preserving 14280 bytes of symbol table [/etc/drivers/rsdd]
  no symbol [/etc/drivers/posixdd]
  no symbol [/etc/drivers/dtropendd]

KDB(4)> stbl //list symbol table entries
  LDRENTRY  TEXT  DATA  TOC  MODULE  NAME
  1 00000000 00000000 00000000 00207EF0 /unix
  2 0B04C400 0156F0F0 015784F0 01578840 /etc/drivers/rsdd

KDB(4)> rmst 2 //ignore second entry

KDB(4)> stbl //list symbol table entries
  LDRENTRY  TEXT  DATA  TOC  MODULE  NAME
  1 00000000 00000000 00000000 00207EF0 /unix

KDB(4)> stbl 1 //list a symbol table entry
  LDRENTRY  TEXT  DATA  TOC  MODULE  NAME
  1 00000000 00000000 00000000 00207EF0 /unix

st_desc addr.... 00153920
  symoff........ 002A9EB8
  nb_sym........ 0000551E

KDB(0)> lke //summary of kernel loader entries
  ADDRESS  FILE  FILESIZE  FLAGS  MODULE  NAME
  1 070E6000 03634EA0 0000ADF8 000080272 random64/usr/lib/drivers/random
  2 070E1000 070E1000 00000F8F 00180248 /unix
  3 070E6000 07541000 00000AFF 000008272 nfs.ext64/usr/lib/drivers/nfs.ext
  4 070E6000 070DF000 00000F8F 00180248 /unix
  5 070E6000 03634A60 00000F30 000008272 nfs_kdes_null.ext64/usr/lib/drivers/nfs_kdes.ext
  6 070E6000 07016000 00000F0D 00180248 /unix
  7 070E6000 036346C0 00000390 000008262 syscalls64.ext64/usr/lib/drivers/syscalls64.ext
  8 070E6000 0362EA60 00000C50 000008272 perfstat64/usr/lib/perf/perfstat
  9 070E6000 070E6000 00000F0D 00380248 /unix
  10 070E6000 0362EA70 00000420 000008262 smt_loadpin64/usr/lib/drivers/smt_loadpin
  11 070E6000 036290DE 00000498 000008272 smt_load64/usr/lib/drivers/smt_load
  12 070E6000 070C0000 00000E40 00180248 /unix
  13 070E6000 03616B00 000002F8 000008272 ptydd64/usr/lib/drivers/ptydd
  14 070E6000 070E8000 000000C0 00180248 /unix

(0)> more (^C to quit) ? ^C //interrupt

KDB(0)> lke 7 //show loader entry, populate cache
  ADDRESS  FILE  FILESIZE  FLAGS  MODULE  NAME

  7 070E6000 036346C0 00000390 000008262 syscalls64.ext64/usr/lib/drivers/syscalls64.ext
```

Chapter 26. Loader subcommands 253
le_filename... 070E68B8 le_file........ 036346C0
le_filesize... 00000390 le_data........ 036349B8
le_tid........ 036349B8 le_datasize.... 00000908
le_usecount... 00000002 le_loadcount... 00000002
le_ndepend.... 00000001 le_maxdepend... 00000001
le_ule........ 00000000 le_deferred.... 00000000
le_exports.... 00000000 le_deferred.... 00000000
le_searchlist.. 00000000 le_dlusecount.. 00000000
le_dllindex... FFFFFFFF le_lex........ 00000000
le_fh......... 00000000 le_depend.... @ 070E68B0
TOC0......... 03634A28

KDB(0)> kde -s 7 //show loader entry, populate cache without printing symbols
ADDRESS  FILE FILESIZE  FLAGS  MODULE NAME
7 070E68B0 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscall s64.ext
le_flags....... TEXT DATA IN TEXT DATA DATA EXISTS 64
le_next........ 070E6900 le_svc_sequence 00000000
le_fp.......... 00000000
le_filename... 070E68B8 le_file........ 036346C0
le_filesize... 00000390 le_data........ 036349B8
le_tid........ 036349B8 le_datasize.... 00000908
le_usecount... 00000002 le_loadcount... 00000002
le_ndepend.... 00000001 le_maxdepend... 00000001
le_ule......... 00000000 le_deferred.... 00000000
le_exports.... 00000000 le_deferred.... 00000000
le_searchlist.. 00000000 le_dlusecount.. 00000000
le_dllindex... FFFFFFFF le_lex........ 00000000
le_fh........ 00000000 le_depend.... @ 070E68B0
TOC0......... 03634A28

KDB(0)> kde -a 070E68B0 //show loader entry by address, populate cache
ADDRESS  FILE FILESIZE  FLAGS  MODULE NAME
070E68B0 036346C0 00000390 00080262 syscalls64.ext64/usr/lib/drivers/syscall s64.ext
le_flags....... TEXT DATA IN TEXT DATA DATA EXISTS 64
le_next........ 070E6900 le_svc_sequence 00000000
le_fp.......... 00000000
le_filename... 070E68B8 le_file........ 036346C0
le_filesize... 00000390 le_data........ 036349B8
le_tid........ 036349B8 le_datasize.... 00000908
le_usecount... 00000002 le_loadcount... 00000002
le_ndepend.... 00000001 le_maxdepend... 00000001
le_ule......... 00000000 le_deferred.... 00000000
le_exports.... 00000000 le_deferred.... 00000000
le_searchlist.. 00000000 le_dlusecount.. 00000000
le_dllindex... FFFFFFFF le_lex........ 00000000
le_fh........ 00000000 le_depend.... @ 070E68B0
TOC0......... 03634A28

KDB(0)> kde -1 //list the cache

KDB(0)> kde -123 //Loader entries for 32-bit shared libraries
ADDRESS  FILE FILESIZE  FLAGS  MODULE NAME
1 F100009A00E6B600 000DCE000 00000491C 00000882 /usr/lib/nls/loc/ucv/UTF32TBL
2 F100009A00EB0500 0017E0000 00026663 00000882 /usr/lib/nls/loc/loc/utf-32_UTF-8
3 F100009A00EB4900 00C7F0C0 0000E73A 000000C9 shr.o/usr/lib/libc6ti.a
4 F100009A00EB3900 00C7F0C0 00006FB2 000000C9 shr.o/usr/lib/libcsm_clog.a

254  KDB Kernel debugger and kdb command
Chapter 26. Loader subcommands

KDB(0)> lke -164 // loader entries for 64-bit shared libraries
   ADDRESS   FILE   FILESIZE   FLAGS   MODULE       NAME
  1 F100009F30049F00  900000000051AC0  0001073D  000800C0  shr_64.o/usr/lib/libcfg.a
  2 F100009F30049E00  900000000045920  0000A898  000800C0  shr_64.o/usr/lib/libdpi20.a
  3 F100009F30049D00  9000000000319C0  000133D1  000800C0  shr_64.o/usr/lib/libsrc.a
  4 F100009F30049C00  90000000001C360  0001488C  000800C0  shr_64.o/usr/lib/libodm.a
  5 F100009F30049B00  900000000063280  00000A2B  000800C0  shr_64.o/usr/lib/libcrypt.a
  6 F100009F30049A00  900000000243000  00223526  000800C0  shr_64.o/usr/lib/libc.a
  7 F100009F30049900  900000000063280  00000A2B  000800C0  shr_64.o/usr/lib/libcrypt.a
  8 F100009F30049800  900000000063280  00000A2B  000800C0  shr_64.o/usr/lib/libcrypt.a
  9 F100009F30049700  900000000045920  0000A898  00080082  shr_64.o/usr/lib/libdpl20.a
 10 F100009F30049600  9000000000319C0  000133D1  00080082  shr_64.o/usr/lib/libbsrc.a
 11 F100009F30049400  900000000051AC0  0001073D  000800C0  shr_64.o/usr/lib/libcrypt.a
 12 F100009F30049500  900000000051AC0  0001073D  000800C0  shr_64.o/usr/lib/libcrypt.a
KDB(0)> lke -p 1 // loader entries for process slot 1
   ADDRESS   FILE   FILESIZE   FLAGS   MODULE       NAME
  1 F00000002FFC8300  004E0000  0002B4EB  000021740 shr_xpg5.o/usr/lib/libpthreads.a
  2 F00000002FFC8200  004A0000  000038C7  000021740 shr_comm.o/usr/lib/libpthreads.a
  3 F00000002FFC8100  007A0F8  000800846  00001740  shr.o/usr/lib/libcrypt.a
  4 F00000002FF3C578  001DE000  001F8008  00001740  shr.o/usr/lib/libc.a
  5 F00000002FF3C4C0  00000000  0000850E  00005242  init
KDB(0)> lke -n syscalls64.ext64 // loader entry by name
   ADDRESS   FILE   FILESIZE   FLAGS   MODULE       NAME
  7 070E6B00  036346C0  00000390  00000826  syscalls64.ext64/usr/lib/drivers/syscalls64.ext
lle subcommand

Purpose
The lle subcommand lists loader entries.

Syntax
lle [-k | -l32 | -l64 | -a addr] [-p slot] [-A] [-v]

Parameters
-k Lists the kernel loader entries.
-l32 Lists the 32-bit library loader entries.
-l64 Lists the 64-bit library loader entries.
-a Lists the loader entry at the specified address.
-p Lists the loader entries for the specified process.
-A Lists the loader anchor information.
-v Lists all fields in the selected entries.
address Specifies the address of a loader entry. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.
slot Specifies a decimal process slot.

Aliases
No aliases.

Example
The following is an example of how to use the lle subcommand:
KDB(0)> lle -k //kernel loader entries
ADDRESS FILE FILESIZE FLAGS MODULE NAME
1 07058000 03634EA0 0000ADF8 00080272 /usr/lib/drivers/random(random64)
2 07172100 07175000 00000FF8 00180248 /unix
3 07058E00 07541000 00000430 00080272 /usr/lib/drivers/nfs_kdes.ext(nfs_kdes_null.ext64)
4 07058900 07173000 00000430 00080272 /unix
5 07058C00 03634A60 00000043 00080272 /usr/lib/drivers/nfs_kdes.ext(nfs_kdes_null.ext64)
6 07058D00 07170000 00000FD0 00180248 /unix
7 07058E00 036346C0 00000390 00080262 /usr/lib/drivers/syscalls64.ext(syscalls64.ext64)
8 07058F00 07170000 00000FD0 00180248 /unix
9 07058D00 07170000 00000FD0 00380248 /unix
10 07058700 0362E7A0 000002A0 00080262 /usr/lib/drivers/smt_loadpin(smt_loadpin64)
11 07058600 035FC940 0001A518 00080272 /usr/lib/drivers/iscsidd(iscsidd64)
12 07058800 07178000 00000EB8 00180248 /unix
13 07058400 0362E7A0 000002A0 00080262 /usr/lib/drivers/smt_loadpin(smt_loadpin64)
14 07058500 0716E000 000000C0 00180248 /unix
15 07058300 0363F940 0001A518 00080262 /usr/lib/drivers/iscsidd(iscsidd64)
16 07058100 036F80E0 00004838 00080272 /usr/lib/drivers/if_en(if_en64)
17 07058200 07160000 000000B8 00180248 /unix
18 07013F00 072A65F0 00000827 00180248 /usr/lib/drivers/isa/msed_chrp(msed_chrp64)
19 07013F00 072A0000 00000827 00180248 /usr/lib/drivers/isa/msed_chrp(msed_chrp64)

(0)> more (^C to quit) ^C //interrupt
KDB(0)> lle -l32 //32-bit library loader entries
ADDRESS FILE FILESIZE FLAGS MODULE NAME
1 F1000000000AD000 00045080 0000000C0 /usr/lib/libc.a(shr.o)
2 F1000000000AD000 00045080 0000000C0 /usr/lib/libc.a(shr3.o)
3 F1000000000AD000 00045080 0000000C0 /usr/lib/libc.a(ansi32.o)
4 F1000000000AD000 00045080 0000000C0 /usr/lib/libc.a(shr.o)
Chapter 26. Loader subcommands 257
le_ndepend..... 00000004  le_maxdepend... 00000004
le_deferred..... 00000000  le_ule........... 00000000
le_exports...... F1000009AE0067000  le_de........... 00000000
le_searchlist... F00000002FFCA080  le_dlusecount... 00000000
le_dindex...... 00000003  le_lex........... 00000000
le_depend...... F10000009AE0049600
                            F00000002FFC8200  /usr/lib/libpthreads.a(shr_comm.o)
                            F00000002FF3C578  /usr/lib/libc.a(shr.o)
                            0701AD00  /unix

2 Loader Entry 0F00000002FFC8200
le_filename.... F1000009AE0049788 /usr/lib/libpthreads.a(shr_comm.o)
le_flags....... DATA LIBEXPORTS DATAEXISTS USEASIS DATAMAPPED
(0)> more (^c to quit) ? ^c //interrupt
KDB(0)> l1e -p 1 -A    //loader anchor information
ANCHOR ADDRESS... F00000002FF3C400
l1_a_loadlist..... F00000002FFC8300
l1_a_flags........ DEFERRED DATA_HEAP
l1_a_lib_le_sid.... 0000B9AB
ldr64............ 00D05160
exp subcommand

Purpose
The exp subcommand looks for an exported symbol or displays the entire export list.

Syntax
exp [symbol]

Parameters
symbol Specifies the symbol name to locate in the export list. This parameter is an ASCII string.

If no parameter is specified, the entire export list is displayed. If a symbol name is specified as a parameter and that symbol is in the export list, then that symbol name is displayed. If a symbol name is specified that is not in the list, then all symbols that begin with the input string are displayed.

Aliases
No aliases.

Example
The following is an example of how to use the exp subcommand:

KDB(0)> exp //list export table
000814D4 pio_assist
01A7708 puthere
007BE90 mmminfo
00081FD4 socket
01A2BA50 tcp_input
01A2BAFC in_pcb_hash_del
01A7BE8 adjmsg
0008AB8 executeit
00325138 loif
019B0874 lvm_kp_tid
00081E6E ns_detach
019A9390 mps_wakeup
01A2C50 ip_forward
00081E60 ksettickd
000810AC uiomove
000811EC blkflush
0018D97C setpriv
01A5CD38 clntkudp_init
00082000 sqoremque
00178824 devstos
00081984 rtinithead
01A5CD8C xdr_rmtcall_args
(0)> more (^C to quit) ? ^C //interrupt
KDB(0)> exp send //display symbol 'send'
007EF0B4 send
KDB(0)> exp sen //display all symbols that start with 'sen'
........ 2573 export entries
007EF54C send_file
007EF078 sendmsg
007EF090 sendto
007F5B38 send_file_duration
007EF0B4 send
KDB(0)>
Chapter 27. Display context information subcommands

The subcommands in this category display context information. These subcommands include the following:

- pnda
- ppda
- mst
- lastbackt
- ttid
- tpid
- ppid
- ptid
- rq
- rq!
- lq
- cr
- svmon
- mem!
- cred
pnda subcommand

Purpose
The pnda subcommand displays the per-node data area pnda structures for each processor.

Syntax
pnda [* | -a | cpu | effectiveaddress]

Parameters
* Displays a summary of the pnda structure for each processor. Multiple processors can share the same pnda structure.
-a Causes the subcommand to display the pnda structure associated with each processor on the system.
cpu Specifies the number of the processor for which you want to display the pnda structure.
effectiveaddress Displays the effective address for which you want to display the pnda structure.

When used without parameters, the pnda subcommand displays the pnda structure for the current processor. With parameters, the pnda subcommand can either display a summary of all pnda structures on the system, or it can display a pnda structure for a specific processor.

Aliases
No aliases.

Example
The following is an example of how to use the pnda subcommand:

KDB(0)> pnda *

<table>
<thead>
<tr>
<th>CPU</th>
<th>SRAD</th>
<th>CPUBITM</th>
<th>MEMPOOL_ON_SRAD</th>
<th>MRQ_SRAD</th>
<th>RSET</th>
<th>ATT_ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>00566B50</td>
<td>0</td>
<td>F000000000000000</td>
<td>0000000000000000</td>
<td>02171000</td>
<td>0040FD50</td>
<td>00566D20</td>
</tr>
<tr>
<td>00566B50</td>
<td>1</td>
<td>F000000000000000</td>
<td>0000000000000000</td>
<td>02171000</td>
<td>0040FD50</td>
<td>00566D20</td>
</tr>
<tr>
<td>00566B50</td>
<td>2</td>
<td>F000000000000000</td>
<td>0000000000000000</td>
<td>02171000</td>
<td>0040FD50</td>
<td>00566D20</td>
</tr>
<tr>
<td>00566B50</td>
<td>3</td>
<td>F000000000000000</td>
<td>0000000000000000</td>
<td>02171000</td>
<td>0040FD50</td>
<td>00566D20</td>
</tr>
</tbody>
</table>

KDB(0)> pnda 00566B50 //pnda address from the first column of previous subcommand

00566B50 | 0 | F000000000000000 | 0000000000000000 | 02171000 | 0040FD50 | 00566D20 |

sradid.................00000000
pndas[0].................00566B50
cpu2srad[0]..............0000 cpu2srad[01]..............0000
cpu2srad[02]..............0000 cpu2srad[03]..............0000
srad2cpu[0].............0000
cpubitm[0]........F0000000000000000
num_cpus_onl[0]........00000004
max_cpus[0]..............00000004
max_num_srads...........00000001 num_srad_onl...........00000001
sys_cpus_onl...........00000004 sys_max_cpus...........00000004
first_srad_with_cpus........00000000
mem_on_srad[0]........0000000000000000
mrq_srad..............02171000 gc_heap..............00000000
srad_rptr..............0040FD50 srad_rset..............00566D18
srad_att_entry...........00566D20
netkmem..............3287D0000
entry.start............0000000000000000 entry.nbytes............00000000

262 KDB Kernel debugger and kdb command
entry.next..............00000000  entry.policy............00000000
entry.cursor............00000000  entry.rset............00566038
KDB(0)>
ppda subcommand

Purpose
The ppda subcommand displays a summary for all ppda structures with the * parameter. Otherwise, details for the current or specified processor are displayed.

Syntax
ppda [* | cpu | effectiveaddress]

Parameters
* Displays a summary for all processors.
cpu Displays the data for the ppda structure for the specified processor. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of a ppda structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Aliases
No aliases.

Example
The following is an example of how to use the ppda subcommand:
KDB(1)> ppda *
SLT CSA CURTHREAD SRR1 SRR0 ppda+000000
0 004ADEB0 thread+000178 40000030 1002DC74 ppda+000390
1 0048EB0 thread+000234 00099030 1d_usecount+00045C ppda+000600
2 004C3EB0 thread+0002F0 00000030 000012F0 ppda+000900
3 004CEEB0 thread+0003AC 00000030 000012F0 ppda+000C00
4 0049EB0 thread+000468 000F0030 000012F0 ppda+000F00
5 004E4EB0 thread+000524 00000030 10019B70 ppda+001200
6 004EE80 thread+0005E0 00000030 000012F0 ppda+001500
7 004FAEB0 thread+00069C 00000030 000012F0
KDB(1)> ppda //current processor data area

Per Processor Data Area [000C0300]
csa......................004B8EB0 mstack..............004B7EB0
fowner..................00000000 curthread.............E6000234
syscall..................0001879B intr..................E0100080
i_softis................00000000 i_softpri.............4000
ppda_pal[0]..............00000000 ppda_pal[1]..............00000000
ppda_pal[2]..............00000000 ppda_pal[3]..............00000000
phy_cpuid................0001 ppda_fp_cr...........28222881
flih save[0]...........00000000 flih save[1]........2FF3B338
flih save[2]...........002E65E0 flih save[3]........00000003
flih save[4]...........00000002 flih save[5]........00000003
flih save[6]...........002E6750 flih save[7]........00000000
dsisr...................40000000 dsi_flag.............00000003
dar.....................2FF9F884
dssave[0]................2FF3B2A0 dssave[1]...........002E65E0
dssave[2]..............00000000 dssave[3]...........002A481C
dssave[4]..............E6001ECB dssave[5]...........00002A33
dssave[6]..............00002A33 dssave[7]...........00000001
dssrr0..................002705AC dssrr1...............00000903
dssprgl................2FF9F880 dscptr.............00000000
dslr.....................002704CC dsxer...............20000000
dsmq.....................00000000 pimapstk...........0021C800
Chapter 27. Display context information subcommands
mst subcommand

Purpose
The mst subcommand prints the Machine State Save Area.

Syntax
mst [slot] [-a] effectiveaddress

Parameters
-a effectiveaddress  Specifies the effective address of a Machine State Save Area to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.
slot  Specifies the thread slot number. This value must be a decimal value.

If a thread slot number is specified, the Machine State Save Area for the specified slot is displayed. If an effective address is entered, it is assumed to be the address of the Machine State Save Area.

Aliases
No aliases.

Example
The following is an example of how to use the mst subcommand:
KDB(0)> mst  //current mst

Machine State Save Area
iar : 0002599C  msr : 00009030  cr : 20000000  lr : 00025988
ctr : 000259EC  xer : 00000000  mq : 00000000
r0 : 00000000  r1 : 2FF3B338  r2 : 002E65E0  r3 : 00000003  r4 : 00000002
r5 : 00000006  r6 : 002E6750  r7 : 00000000  r8 : DEADBEEF  r9 : DEADBEEF
r10 : DEADBEEF  r11 : 00000000  r12 : 00000030  r13 : DEADBEEF  r14 : DEADBEEF
r15 : DEADBEEF  r16 : DEADBEEF  r17 : DEADBEEF  r18 : DEADBEEF  r19 : DEADBEEF
r20 : DEADBEEF  r21 : DEADBEEF  r22 : DEADBEEF  r23 : DEADBEEF  r24 : DEADBEEF
r25 : DEADBEEF  r26 : DEADBEEF  r27 : DEADBEEF  r28 : 000034E0  r29 : 000C6158
r30 : 0000C078  r31 : 00005004
s0 : 00000000  s1 : 007FFFFF  s2 : 0000F00F  s3 : 007FFFFF  s4 : 007FFFFF
s5 : 007FFFFFFF  s6 : 007FFFFFFF  s7 : 007FFFFFFF  s8 : 007FFFFFFF  s9 : 007FFFFFFF
s10 : 007FFFFFFF  s11 : 007FFFFFFF  s12 : 007FFFFFFF  s13 : 0000C00C  s14 : 00004004
s15 : 007FFFFFFF
prev  00000000  kjmpbuf  00000000  stackfix  00000000  intpri  0B
curid  00000306  sralloc  E01E0000  ioalloc  00000000  backt  00
flags  00 tid  00000000  excp_type  00000000
fpscr  00000000  fpeu  00  fpinfo  00  fpscrx  00000000
o_iar  00000000  o_arg1  00000000  o_vaddr  00000000  mtext  00000000
Except :
csr  2FEC6B78  dsisr  40000000  bit set: DSISR_PFT
sval  000019DD  dar  2FEC6B78  dsisr  00000000
KDB(0)> mst 1  ///slot 1 is thread+0000A0

Machine State Save Area
iar : 00038E00  msr : 00001030  cr : 2A442424  lr : 00038E00
ctr : 002BC000  xer : 00000000  mq : 00000000
r0 : 60017017  r1 : 2FF3B330  r2 : 002E65E0  r3 : 00000000  r4 : 00000002
r5 : E000000C  r6 : 00000109  r7 : 00000000  r8 : 00000300  r9 : 00000001
r10 : 2FF3B330  r11 : 00000000  r12 : 00001030  r13 : 00000001  r14 : 2FF22F5C
r15 : 2FF22F5C  r16 : DEADBEEF  r17 : DEADBEEF  r18 : 0000040F  r19 : 00000000
r20 : 00000000  r21 : 00000003  r22 : 01000001  r23 : 00000001  r24 : 00000000
r25 : E000014C  r26 : 00001A0B  r27 : 00000000  r28 : E3000160  r29 : E60000BC
Chapter 27. Display context information subcommands
**lastbackt subcommand**

**Purpose**

The `lastbackt` subcommand prints the context (Machine State Save Area) for when the last backtracking fault was taken on either the current processor or the specified processor.

**Syntax**

`lastbackt [cpu]`

**Parameters**

`cpu` Specifies a cpu index as a decimal value. If the cpu index is omitted, `lastbackt` defaults to the current cpu context.

**Aliases**

No aliases.

**Example**

The following is an example of how to use the `lastbackt` subcommand:

```
KDB(0)> lastbackt //use current cpu context
```

Machine State Save Area

```
 iar : 0002599C msr : 00009030 cr : 20000000 lr : 00025988
ctr : 00025BEC xer : 00000000 mq : 00000000
r0 : 00000000 r1 : 2FF3B338 r2 : 002E65E0 r3 : 00000003 r4 : 00000002
r5 : 00000006 r6 : 002E6750 r7 : 00000000 r8 : DEADBEEF r9 : DEADBEEF
r10 : DEADBEEF r11 : 00000000 r12 : 00000930 r13 : DEADBEEF r14 : DEADBEEF
r15 : DEADBEEF r16 : DEADBEEF r17 : DEADBEEF r18 : DEADBEEF r19 : DEADBEEF
r20 : DEADBEEF r21 : DEADBEEF r22 : DEADBEEF r23 : DEADBEEF r24 : DEADBEEF
r25 : DEADBEEF r26 : DEADBEEF r27 : DEADBEEF r28 : 000034E0 r29 : 000C6158
r30 : 000C0578 r31 : 00005004
s0 : 00000000 s1 : 007FFFFF s2 : 0000F000 s3 : 007FFFFF s4 : 007FFFFF
s5 : 007FFFFF s6 : 007FFFFF s7 : 007FFFFF s8 : 007FFFFF s9 : 007FFFFF
s10 : 007FFFFF s11 : 007FFFFF s12 : 007FFFFF s13 : 000C00C s14 : 00000404
s15 : 007FFFFF
prev 00000000 kjmbuf 00000000 stackfix 00000000 intpri 0B
curid 000030e6 slralloc E01E0000 alloclc 00000000 backt 03
flags 00 tid 00000000 excp_type 00000000
fpscr 00000000 fpueu 00 fpinfo 00 fpscrx 00000000
o_iar 00000000 o_arg1 00000000 o_argl 00000000
excbranch 00000000 o_vaddr 00000000 mstext 00000000
```

Except:

```
csr 2FEC6B78 dsir 40000000 bit set: DSISR_PFT
sval 000019DD dar 2FEC6B78 dsir 00001006
```

```
KDB(0)> lastbackt 1 //use cpu 1
```

Machine State Save Area

```
 iar : 00038ED0 msr : 00001030 cr : 2A442424 lr : 00038ED0
ctr : 002BCC00 xer : 00000000 mq : 00000000
r0 : 60017017 r1 : 2FF3B300 r2 : 002E65E0 r3 : 00000000 r4 : 00000002
r5 : 0E000000 r6 : 00000109 r7 : 00000000 r8 : 00000000 r9 : 00000000
r10 : 2FF3B300 r11 : 00000000 r12 : 00001030 r13 : 00000001 r14 : 2FF22F54
r15 : 2FF22F5C r16 : DEADBEEF r17 : DEADBEEF r18 : 000040F0 r19 : 00000000
r20 : 00000000 r21 : 00000003 r22 : 01000001 r23 : 00000001 r24 : 00000000
r25 : 0E000014C r26 : 00001A08 r27 : 00000000 r28 : E300160 r29 : E600008C
r30 : 00000004 r31 : 00000004
s0 : 00000000 s1 : 007FFFFF s2 : 0000A00A s3 : 007FFFFF s4 : 007FFFFF
s5 : 007FFFFF s6 : 007FFFFF s7 : 007FFFFF s8 : 007FFFFF s9 : 007FFFFF
```

268  KDB Kernel debugger and kdb command
Chapter 27. Display context information subcommands
ppid subcommand

Purpose
The ppid subcommand displays the process entry belonging to a process.

Syntax
ppid [\[-h \-d \]pid]

Parameters
-h Specifies that the pid parameter is in hexadecimal.
-d Specifies that the pid parameter is in decimal.
pid Specifies the process ID for which you want to display the process entry. This value must be a decimal or a hexadecimal value as specified by the \[-h \-d \] options, or as required by the \texttt{hexadecimal\_wanted} toggle specified with the \texttt{set} subcommand. If no process ID is specified, the process table entry for the current process is displayed.

Aliases
pr\_pid

Example
The following is an example of how to use the ppid subcommand:

KDB(0)> ppid 03C88
SLOT NAME STATE PID PPID PGRP UID ADSPACE CL #THS

pvproc+007800 60 ksh ACTIVE 03C88 00001 03C88 00000 000E01BC 0 0001

KDB(0)> ppid -d 6996
SLOT NAME STATE PID PPID PGRP UID ADSPACE CL #THS

pvproc+003600 27 errdemone ACTIVE 01B54 00001 01B54 00000 00008161 0 0001

...
**ptid subcommand**

**Purpose**
The **ptid** subcommand displays the process entry corresponding to a thread.

**Syntax**
`ptid [-h | -d] tid`

**Parameters**
- `-h` Specifies that the tid parameter is in hexadecimal.
- `-d` Specifies that the tid parameter is in decimal.

`tid` Specifies the thread ID for which you want to display the process entry. This value must be a decimal or a hexadecimal value as specified by the `-h` or `-d` options, or as required by the `hexadecimal_wanted` toggle specified with the `set` subcommand. If no thread ID is specified, the process table entry for the current thread is displayed.

**Aliases**
`pr_tid`

**Example**
The following is an example of how to use the **ptid** subcommand:

```
KDB(0)> th -n ksh
SLOT NAME STATE TID PRI RQ CPUID CL WCHAN
pvthread+003100 98 ksh SLEEP 0062D5 03D 1 0 70142890
KDB(0)> ptid 0062D5
SLOT NAME STATE PID PPID PGRP UID ADSPACE CL #THS
pvproc+007800 60 ksh ACTIVE 03C88 00001 03C88 00000 00E01BC 0 0001
```

```
NAME....... ksh
STATE...... stat :07 .... xstat :0000
FLAGS...... flag :00200001 LOAD EXECED
............. flag2 :00000000
............. atomic :00040000 ORPHANPGRP
LINKS...... child :00000000
............. siblings :E2005400 <pvproc+005400>
............. uidinfo :0055C240
............. ganchor :E2007800 <pvproc+007800>
THREAD..... threadlist :EA003100 <pvthread+003100>
DISPATCH... synch :FFFFFFFF
AACCT...... projid :00000000 ............ sprojid :00000000
............. subproj :0000000000000000
............. file id :0000000000000000000000000000000000000000
............. flags :0000
WLM........ class/wlm :00/0000
............. time of SIGTERM:00000000
............. wlm_nvpages :0000000000000000 0
... <The output here is identical to the "p" or "proc" command, and is for all other examples here>
```

```
KDB(0)> ptid -d 17923
SLOT NAME STATE PID PPID PGRP UID ADSPACE CL #THS
pvproc+004C00 38 sendmail ACTIVE 02674 01762 02674 00000 000881B7 0 0001
```

```
NAME....... sendmail
STATE...... stat :07 .... xstat :0000
```
FLAGS...... flag :00200001 LOAD EXEED
............ flag2 :00000000
............ atomic :00040000 ORPHANGRP
....
ttid subcommand

Purpose
The ttid subcommand displays the thread table entry for a specific thread.

Syntax
```
ttid [-h | -d] tid
```

Parameters
```
-h Specifies that the tid parameter is in hexadecimal.
-d Specifies that the tid parameter is in decimal.
tid Specifies the thread ID. This value must be a decimal or a hexadecimal value as specified by the [-h or -d] options, or as required by the hexadecimal_wanted toggle specified with the set subcommand. If no thread ID is specified, the entry for the current thread is displayed.
```

Aliases
```
th_tid
```

Example
The following is an example of how to use the ttid subcommand:
```
KDB(4)> p //print process table
SLOT NAME STATE PID PPID PGRP UID EUID ADSPACE
... proc+000100 1 init ACTIVE 00001 00000 00000 00000 00000A005
... proc+000C00 12 gil ACTIVE 00C18 00000 00000 00000 00026013
...
KDB(4)> tpid 1 //print thread(s) of process pid 1
SLOT NAME STATE TID PRI CPUID CPU FLAGS WCHAN
thread+0000C0 1 init SLEEP 001D9 03C 000 00000400
KDB(4)> ttid 001D9 //print thread with tid 0x1d9
SLOT NAME STATE TID PRI CPUID CPU FLAGS WCHAN
thread+0000C0 1 init SLEEP 001D9 03C 000 00000400
```

NAME................ init
FLAGS................ WAKEONSIG
WTY....WEVENT
.................stackp64 :00000000 ...............stackp :2FF22DC0
.................state :00000003 ...............wtype :00000001
.................suspend :00000001 ...............flags :00000400
.................atomic :00000000
DATA..............
.................proc :E3000100
.................userp :2FF3B6C0 <__ublock+0002C0>
.................uthreadp :2FF3B400 <__ublock+000000>
THREAD LINK........
.................prevthread :E60000C0
.................nextthread :E60000C0
SLEEP LOCK........
.................unlock84 :00000000 ...............unlock :00000000
.................wchan :00000000 ...............wchan1 :00000000
.................wchan1sid :00000000 ...............wchanoffset :01AB5A58
.................wchan2 :00000000 ...............swchan :00000000
.................eventlist :00000000 ...............result :00000000
.................polevel :000000AF ...............pevent :00000000
.................wevent :00000004 ...............slist :00000000
lockcount :00000000

Dispatch.........
ticks :00000000 prior :E60000C0
next :E60000C0 synch :FFFFFFFC
dispct :000008F6 fpuct :00000000

Scheduler.........
cpu :FFFFFFFC scpuid :FFFFFFFC
affinity :00000001 pri :0000003C
policy :00000000 cpu :00000000
lockpri :0000003D wakepri :0000007F
time :000000FF sav_pri :0000003C

Signal...........
cursig :00000000
(sig pending)
sig :
sigmask :
scp64 :00000000 scp :00000000

Misc................
graphics :00000000 cancel :00000000
lockowner :E60042C0 boosted :00000000
tsleep :FFFFFFFC
userdata64 :00000000 userdata :00000000

274 KDB Kernel debugger and kdb command
tpid subcommand

Purpose
The tpid subcommand displays all thread entries belonging to a process.

Syntax
tpid [-h | -d] pid

Parameters
-h Specifies that the pid parameter is in hexadecimal.
-d Specifies that the pid parameter is in decimal.
 pid Specifies the process ID for which you want to display thread entries. This value must be a decimal or a hexadecimal value as specified by the -h or -d options, or as required by the hexadecimal_wanted toggle specified with the set subcommand. If no process ID is specified, all thread table entries for the current process are displayed.

Aliases
th_pid

Example
The following is an example of how to use the tpid subcommand:

KDB(4)> p * //print process table
SLOT NAME STATE PID PPID PGRP UID EUID ADSPACE
... proc+00100 1 init ACTIVE 00001 00000 00000 00000 00000A05
... proc+00C00 12 g1l ACTIVE 00C18 00000 00000 00000 00026D13
... KDB(4)> tpid 1 //print thread(s) of process pid 1
SLOT NAME STATE TID PRI CPUID CPU FLAGS WCHAN
thread+000C0 1 init SLEEP 00ID9 03C 0000000400
KDB(4)> tpid 00C18 //print thread(s) of process pid 0xC18
SLOT NAME STATE TID PRI CPUID CPU FLAGS WCHAN
thread+000900 12 g1l SLEEP 00C19 025 000001004
thread+000C00 16 g1l SLEEP 01021 025 00000 00000004 netisr_servers+00000
thread+000B40 15 g1l SLEEP 00F1F 025 00000 00000004 netisr_servers+00000
thread+000A80 14 g1l SLEEP 00E1D 025 00000 00000004 netisr_servers+00000
thread+0009C0 13 g1l SLEEP 00D1B 025 00000 00000004 netisr_servers+00000

Chapter 27. Display context information subcommands
rq subcommand

Purpose
The rq subcommand lists threads currently queued on the system run queues.

Syntax
rq [ bucket | effectiveaddress]

Parameters
bucket Lists all threads queued in a particular bucket across all run queues. The bucket is equal to the thread priority minus 1.
effectiveaddress Lists all threads queued in the bucket specified by the effective address.

If the rq subcommand is used with no parameters, a list of all buckets currently containing threads across all run queues is generated. If the rq subcommand is used with parameters, you can restrict the generated list to a particular run queue or to a particular bucket across all run queues.

Aliases
runq

Example
The following is an example of how to use the rq subcommand:
KDB(0)> rq

<table>
<thead>
<tr>
<th>RQ</th>
<th>BUCKET</th>
<th>HEAD</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>02172D04</td>
<td>256</td>
<td>pythread+000100</td>
<td>1</td>
</tr>
<tr>
<td>02172504</td>
<td>256</td>
<td>pythread+000180</td>
<td>1</td>
</tr>
<tr>
<td>02173A1C</td>
<td>70</td>
<td>pythread+005580</td>
<td>1</td>
</tr>
<tr>
<td>02173D04</td>
<td>256</td>
<td>pythread+000200</td>
<td>1</td>
</tr>
<tr>
<td>02173504</td>
<td>256</td>
<td>pythread+000280</td>
<td>1</td>
</tr>
</tbody>
</table>

KDB(0)> rq 02173A1C //bucket address from the RQ column

LOCAL RUNQ( 2) ENTRY( 70) 02173A1C

SLOT NAME | STATE | TID PRI | RQ | CPUID | CL | WCHAN
----------|-------|---------|----|-------|----|-------
 pvthread+005580 | 171>bash | RUN | 00AB67 045 | 2 | 0 |
pvthread+004000 | 154>bash | RUN | 009A7F 045 | 2 | 0 |
pvthread+006100 | 194>bash | RUN | 00C287 045 | 2 | 0 |
pvthread+006500 | 202>bash | RUN | 00CAC9 045 | 2 | 0 |
pvthread+004C00 | 152>bash | RUN | 009851 045 | 2 | 0 |
pvthread+006380 | 199>bash | RUN | 00C701 045 | 2 | 0 |
pvthread+006280 | 197>bash | RUN | 00C5B7 045 | 2 | 0 |

KDB(0)> rq 256 //bucket number from the RQ column

LOCAL RUNQ( 0) ENTRY(256) 02172D04

SLOT NAME | STATE | TID PRI | RQ | CPUID | CL | WCHAN
----------|-------|---------|----|-------|----|-------
 pvthread+000100 | 2>wait | RUN | 000205 OFF | 0 | 00000 | 0 |

LOCAL RUNQ( 1) ENTRY(256) 02172504

SLOT NAME | STATE | TID PRI | RQ | CPUID | CL | WCHAN
----------|-------|---------|----|-------|----|-------
 pvthread+000180 | 3>wait | RUN | 000307 OFF | 1 | 00001 | 0 |

LOCAL RUNQ( 2) ENTRY(256) 02173D04

SLOT NAME | STATE | TID PRI | RQ | CPUID | CL | WCHAN
----------|-------|---------|----|-------|----|-------
 pvthread+000200 | 4>wait | RUN | 000409 OFF | 2 | 00002 | 0 |

LOCAL RUNQ( 3) ENTRY(256) 02173504
pvthread+000280  5>wait  RUN  00050B  OFF  3  00003  0
GLOBAL RUNQ(node  0) ENTRY(256) 02171904
KDB(0)>
**rq** subcommand

**Purpose**
The *rq* subcommand displays information about run queues on the system.

**Syntax**
`rq [ -mrq | queue | slot ]`

**Parameters**
- `-mrq` Displays information about all *mrq* nodes in the system.
- `queue` Specifies the effective address for the run queue structure specified by the effective address.
- `slot` Specifies the run queue structure you want to display.

If the *rq* subcommand is run without any parameters, a summary line for each run queue in the system is displayed. If the *rq* subcommand is run with parameters, a specific run queue structure or the *mrq* nodes in the system are displayed.

**Aliases**
- `rqa`

**Example**
The following is an example of how to use the *rq* subcommand:

```
KDB(0)> rq -mrq
primary_grq............ 2171400
run_queue_max_local...... 00000003 run_queue_max_global..... 00000080
num_nodes_onl........... 00000001 nodep @ 11EA710
MRQ_NODE @ 2171000
my_ndx............ 0000 rq_start_ndx..... 0000 lbolt....... 0006
active_rqs......... 0004 max_rqs......... 0004
rq_mask............ F0000000 00000000 00000000 00000000
S2_threshold........ 0000 num_S2........... 0001 S3_threshold. 00000180
thread_count...... 00A3 load............ 00000003 rq_slot..... @ 21711C8
sched_tid........... 00000003 reaper_tid..... 00000000
zstart............ 0 zfinal............ E200D000
pref_S2id......... 0 S2_stealable.... 0 0 0 S2id............ 0 0
S3_anysteals..... 0 S1_loads........... 00000000 00000000 00000000 00000000
KDB(0)>
```

```
KDB(0)> rq
RQ Node CPUs First Threads st1 ustl any S1st1 S2st1 S3st1 Busy Load

     0 0 1 0 38 0 0 0 0.0 1.1 0.0 0 0.0
     1 0 1 1 44 0 0 0 0.0 0.5 0.0 0 0.0
     2 0 1 2 42 0 0 0 0.0 0.0 0.0 0 0.0
     3 0 1 3 39 0 0 0 0.0 0.0 0.0 0 0.0
     4 0 1 4 36 0 0 0 0.0 0.0 0.0 0 0.0
     5 0 1 5 33 0 0 0 0.0 0.0 0.0 0 0.0
     0 0 4 0 0 0 0 0 0.0 0.0 0.0 0 0.0
KDB(0)> rq 3 //slot number from RQ column in rq subcommand
RUN_QUEUE @ 2173000
runrun............ 00000000 rq_stealable.... 00000000 S2_stealable... 00
rq_unstealable.... 00000000 rq_load............ 00000000 rq_S2id........ 00
```

278  KDB Kernel debugger and kdb command
Chapter 27. Display context information subcommands 279

rq_my_node_ndx......... 0000 rq_Slid................. 0003
rq_my_ndx............... 0003 rq_my_node_offset..... 0003
rq_cpu_start_ndx........ 0003 rq_cpu_node_offset.... 0003
rq_active_cpus........... 0001 rq_max_cpus.......... 0001
rq_next_cpu............. 0000
rq_cpus_mask............. 00000000 00000000 00000000 00000000
rq_thread_count.......... 0000000027 rq_node_pointer.... 2171000
rq_busy_ticks............ 0000 rq_busy.............. 0000 rq_tload........ 0000
rq_best_run_pri/fixed.. FF/0 run_queue_lock..... 0 placement_load..... F
rq_steals_this_tick.... 0000 0000 0000 0000
rq_steals_this_second.. 0000 0000 0000 0000
placement_load..... F
dispct 00787334 S0_misses 000006EB S1_misses 000006EB S2_misses 00000000
rq_lbolt................. 0052 rq_curthread_band..... 0000 stealing_active... 00
run_mask[0]............. 00000000 00000000 00000000 00000000
run_mask[4]............. 00000000 00000000 00000000 00000000
shared_S0.............. 00000000 00000000 00000000 00000000
shared_S1.............. 00000000 00000000 00000000 00000000
shared_S2.............. E0000000 00000000 00000000 00000000
thread_run........... 02173108
stealing_blocked... 00000000 00000000 00000000 00000000
banded_load[00]......... 00000000 00000000 00000000 00000000
banded_load[04]......... 00000000 00000000 00000000 00000000
banded_load[08]......... 00000000 00000000 00000000 00000000
banded_load[12]......... 00000000 00000000 00000000 00000000
banded_load_avg[00]..... 00000000 00000000 00000000 00000000
banded_load_avg[04]..... 00000000 00000000 00000000 00000000
banded_load_avg[08]..... 00000000 00000000 00000000 00000000
banded_load_avg[12]..... 00000000 00000000 00000000 00000000
KDB(0)>
Iq subcommand

Purpose
The iq subcommand displays information about threads waiting on a lock.

Syntax
iq [ bucket | effectiveaddress ]

Parameters
bucket Displays information about a thread in the specified lock queue bucket.
effectiveaddress Displays information about a thread in the lock queue bucket that is specified by the effective address.

When run without any parameters, this subcommand displays a list of all threads which are currently waiting on some lock. With a parameter, the subcommand displays information about a waiting thread in a specific lock queue bucket.

Aliases
lockq

Example
The following is an example of how to use the iq subcommand:
KDB(0)> iq

<table>
<thead>
<tr>
<th>BUCKET</th>
<th>HEAD</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>slist_table+0007E0</td>
<td>253 pvthread+003000</td>
<td>1</td>
</tr>
</tbody>
</table>

KDB(0)> iq 253 (lock queue bucket from the previous command)
SLIST_TABLE ENTRY(253): slist_table+0007E0

<table>
<thead>
<tr>
<th>SLOT</th>
<th>NAME</th>
<th>STATE</th>
<th>TID</th>
<th>PRI</th>
<th>RQ</th>
<th>CPUID</th>
<th>CL</th>
<th>WCHAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvthread+003000</td>
<td>96*v3fshelp</td>
<td>SLEEP</td>
<td>006023</td>
<td>03E</td>
<td>2</td>
<td>0 inodes+3F48A64</td>
<td>slist_table+0007E0</td>
<td></td>
</tr>
</tbody>
</table>

KDB(0)>
cr subcommand

Purpose
The cr subcommand displays information about the checkpoint and the restart identifiers from the global crid_table.

Syntax
```
cr [ * | -i id | slot | effectiveaddress ]
```

Parameters
* Causes the crid subcommand to display a summary of all crid structures in the system.
* -i Specifies the checkpoint or restart identifier (CRID) of the crid structure to be displayed.
* slot Specifies the slot number within the crid_table of the crid structure to be displayed.
* effectiveaddress Specifies the effective address of a particular crid structure to be displayed.

If the cr subcommand is run without any parameters, the crid structure is displayed for the current process if one exists. If the cr subcommand is run with parameters, a summary of all crid structures in the table are displayed or any specific crid structure is displayed.

Aliases

crid

Example
The following is an example of how to use the cr subcommand:
```
KDB(0)> cr 42
ADDRESS SLOT ID FLAGS OWNER CHKSYNCH
F10010F00406BA80 42 00000001 00000000 00000000 00000000
ID.......... rcrid :00000001 vcrid :00000000
FLAGS........ flags :00000000
OWNER........ owner :00000000
VIRTUALS... lvpid :0000000000000000
............. lvtid :0000000000000000
............. lvseq :00
CHECKPOINT: chksynch :0000000000000000
............. chkfile :0000000000000000
MEMBERS.... procpv :0000000000000000
KDB(0)>
```
svmon subcommand

Purpose
The svmon subcommand displays information about the memory and paging space use on a per-process basis.

Syntax
svmon [-p pid | -s slot | -a effectiveaddress | * | - ]

Parameters
- p pid Displays detailed information about the process specified by its process identifier.
- s slot Displays detailed information about the process in the specified process slot.
- a effectiveaddress Displays detailed information about the process specified by the effective address of its pvproc structure.
* Displays a brief summary about all the processes on the system when the asterisk ( * ) is the only parameter.
– Displays detailed information about all the processes on the system when the minus sign ( – ) is the only parameter.

When run without any parameters, the svmon subcommand displays information about the memory and paging space use of the running process on the current processor. With parameters, information about other processes or a brief summary of all processes can be displayed.

Aliases
No aliases.

Example
The following is an example of how to use the svmon subcommand:

(0)> svmon

<table>
<thead>
<tr>
<th>Pid</th>
<th>Command</th>
<th>Inuse</th>
<th>Pin</th>
<th>Pgsp</th>
<th>Virtual</th>
<th>64-bit</th>
<th>Mthrds</th>
<th>LPage</th>
<th>Kproc</th>
<th>Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>8196</td>
<td>wait</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 work kernel segment</td>
<td></td>
<td>-</td>
<td>6127</td>
<td>3762</td>
<td>0</td>
<td>6127</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7003</td>
<td>FFFFFFFF work application stack</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5002</td>
<td>F00000002 work process private</td>
<td>-</td>
<td>11</td>
<td>8</td>
<td>0</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(0)> svmon *

<table>
<thead>
<tr>
<th>Pid</th>
<th>Command</th>
<th>Inuse</th>
<th>Pin</th>
<th>Pgsp</th>
<th>Virtual</th>
<th>64-bit</th>
<th>Mthrds</th>
<th>LPage</th>
<th>Kproc</th>
<th>Uid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>swapper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>init</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6127</td>
<td>3762</td>
<td>0</td>
</tr>
<tr>
<td>8196</td>
<td>wait</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12294</td>
<td>wait</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16392</td>
<td>wait</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20490</td>
<td>wait</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24588</td>
<td>reaper</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28686</td>
<td>1rud</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32784</td>
<td>xmdetd</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36882</td>
<td>vmpactr</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40980</td>
<td>pilegc</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45078</td>
<td>xmgc</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49176</td>
<td>netm</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

282 KDB Kernel debugger and kdb command
<table>
<thead>
<tr>
<th>Command</th>
<th>PID</th>
<th>Sched PRI</th>
<th>Time CPU</th>
<th>Usage CPU%</th>
<th>Priv CPU%</th>
<th>Session Access</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gil</td>
<td>6163</td>
<td>3774</td>
<td>0</td>
<td>6163</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>wmsched</td>
<td>6141</td>
<td>3770</td>
<td>0</td>
<td>6141</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>aixmibd</td>
<td>8188</td>
<td>3766</td>
<td>0</td>
<td>8116</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>cron</td>
<td>8082</td>
<td>3766</td>
<td>0</td>
<td>8061</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>random</td>
<td>6141</td>
<td>3770</td>
<td>0</td>
<td>6141</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>jfsz</td>
<td>6141</td>
<td>3770</td>
<td>0</td>
<td>6141</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>dog</td>
<td>6158</td>
<td>3774</td>
<td>0</td>
<td>6158</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>srcmstr</td>
<td>8093</td>
<td>3766</td>
<td>0</td>
<td>8080</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>errdemon</td>
<td>8256</td>
<td>3766</td>
<td>0</td>
<td>8154</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>lvmbb</td>
<td>6141</td>
<td>3770</td>
<td>0</td>
<td>6141</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>kbiod</td>
<td>6146</td>
<td>3771</td>
<td>0</td>
<td>6146</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>syncd</td>
<td>8161</td>
<td>3779</td>
<td>0</td>
<td>8159</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>snmpmibd64</td>
<td>6738</td>
<td>3769</td>
<td>0</td>
<td>6678</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>portmap</td>
<td>8107</td>
<td>3766</td>
<td>0</td>
<td>8095</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>sendmail</td>
<td>8188</td>
<td>3769</td>
<td>0</td>
<td>8116</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>shlap64</td>
<td>6643</td>
<td>3769</td>
<td>0</td>
<td>6633</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rtcmd</td>
<td>6146</td>
<td>3771</td>
<td>0</td>
<td>6146</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>syslogd</td>
<td>8063</td>
<td>3766</td>
<td>0</td>
<td>8052</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rmcd</td>
<td>8435</td>
<td>3766</td>
<td>0</td>
<td>8339</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>hostmibd</td>
<td>8106</td>
<td>3766</td>
<td>0</td>
<td>8087</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>inetd</td>
<td>8069</td>
<td>3766</td>
<td>0</td>
<td>8059</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>muxatmd</td>
<td>8128</td>
<td>3766</td>
<td>0</td>
<td>8116</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rpc.lockd</td>
<td>8048</td>
<td>3766</td>
<td>0</td>
<td>8046</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rpc.statd</td>
<td>8206</td>
<td>3769</td>
<td>0</td>
<td>8185</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>ksh</td>
<td>8161</td>
<td>3766</td>
<td>0</td>
<td>8106</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>biod</td>
<td>8002</td>
<td>3766</td>
<td>0</td>
<td>8001</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IBM.AuditRMd</td>
<td>8359</td>
<td>3775</td>
<td>0</td>
<td>8306</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>diagd</td>
<td>8069</td>
<td>3766</td>
<td>0</td>
<td>8060</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>qdaemon</td>
<td>8039</td>
<td>3766</td>
<td>0</td>
<td>8023</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>writesrv</td>
<td>8040</td>
<td>3766</td>
<td>0</td>
<td>8035</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>uprintfd</td>
<td>7997</td>
<td>3766</td>
<td>0</td>
<td>7995</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rpc.lockd</td>
<td>6185</td>
<td>3779</td>
<td>0</td>
<td>6185</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>IBM.ServiceRM</td>
<td>8285</td>
<td>3774</td>
<td>0</td>
<td>8261</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>IBM.Ermd</td>
<td>8467</td>
<td>3775</td>
<td>0</td>
<td>8406</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>kdb_64</td>
<td>7392</td>
<td>3769</td>
<td>0</td>
<td>6935</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IBM.CSMAgentR</td>
<td>8453</td>
<td>3777</td>
<td>0</td>
<td>8395</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

(0)>
meml subcommand

Purpose
The meml subcommand displays information about the memory lock entries.

Syntax
meml [-l] [-e] effectiveaddress

Parameters
- Specifies the address of a memory lock entries list.
-e Specifies the address of a memory lock entry.
effectiveaddress Identifies the effective address. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Aliases
memlock

Example
The following is an example of how to use the meml subcommand:
KDB(0)> meml?
MEML usage: meml [-l|-e] addr[?] :
: meml -l to print a memlock list
: meml -e to print a memlock list entry
KDB(0)> meml -l 3007A5C0
Memlock list, address 3007A5C0
Memlock list entry, address 3007A5C0
next entry (next) : 000000003007AF60
previous entry (prev) : 0000000000000000
start address (start) : 0000000020000000
number of bytes (size) : 0000000000011000

Memlock list entry, address 3007AF60
next entry (next) : 0000000000000000
previous entry (prev) : 000000003007A5C0
start address (start) : 000000002DF22000
number of bytes (size) : 0000000002001000
KDB(0)> meml -e 000000003007A5C0

Memlock list entry, address 3007A5C0
next entry (next) : 000000003007AF60
previous entry (prev) : 0000000000000000
start address (start) : 0000000020000000
number of bytes (size) : 0000000000011000

284 KDB Kernel debugger and kdb command
cred subcommand

Purpose
The cred subcommand displays the credentials structure for a specific effective address.

Syntax
cred [effectiveaddress]

Parameters

effectiveaddress    Specifies the effective address of a credentials structure.

Aliases
No aliases.

Example
The following is an example of how to use the cred subcommand:

KDB(0)> cred F100660008D42AFC
ref...........00000017 ruid...........00000000 uid...........00000000
suid.........00000000 luid...........00000000 acctid......00000000
gid..........00000000 rgid...........00000000 sgid...........00000000
ngrps.......00000007 pag[0]......00000000
groups[00].00000000 groups[01].00000000 groups[02].00000000
groups[03].00000007 groups[04].00000008 groups[05].0000000A
groups[06].0000000B
pag[01]..F100660000000000 pag[02].0000000000000000
pag[03].0000000000000000 pag[04].0000000000000000
pag[05].0000000000000000 pag[06].0000000000000000
pag[07].0000000000000000 pag[08].0000000000000000
mpriv........FFFFFFFFFF FFFFFFFFFFF lpriv........FFFFFFFFFF FFFFFFFFFFF
epriv........FFFFFFFFFF FFFFFFFFFFF bpriv........FFFFFFFFFF FFFFFFFFFFF
ecap..........00000000 00000000 icap..........00000000 00000000
pcap..........00000000 00000000
KDB(0)>
Chapter 28. Display storage subsystem information subcommands

The subcommands in this category display storage subsystem information. These subcommands include the following:

- “pbuf subcommand” on page 288
- “volgrp subcommand” on page 289
- “pvvol subcommand” on page 291
- “lvvol subcommand” on page 292
- “scd subcommand” on page 293
**pbuf subcommand**

**Purpose**
The *pbuf* subcommand prints physical buffer information.

**Syntax**
`pbuf [effectiveaddress]`

**Parameters**

*effectiveaddress*  
Specifies the effective address of the physical buffer. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the *pbuf* subcommand:

```
KDB(0)> pbuf 34D6A000
pbuf......... 34D6A000
pb............ θ 34D6A000 flags........... 000C0010
      SPLIT MPSAFE INITIAL
forw.......... 000FB505 back............. 00000000
av_forw........ 35776400 av_back........... 00000000
iodone: vm_pfend+000000
vp............ 00000000 dev.............. 000A0003
blkno.......... 0000870 bcount.......... 00001000
error.......... 00000000 resid.......... 00001000
work........... 00000000 options......... 00000000
event.......... 00000000 start.tv_sec.... 403283C3
start.tv_nsec... 00000000
... 00000000 pb_lbuf........... 00000000
pb_sched........ 00000000 pb_pvold........ 00000000
pb_bad.......... 00000000 pb_start........ 00000000
pb_mirror........ 00000000 pb_miravoid..... 00000000
pb_mirbad........ 00000000 pb_mirdone...... 00000000
pb_swriter........ 00000000 pb_type........ 00000000
pb_bbfixtype.... 00000000 pb_bbop......... 00000000
pb_bbstat....... 00000000 pb_whl_stop..... 00000000
pb_part........ 00000000 pb_bbcount...... 00000000
stripe_next...... 00000000 stripe_status.... 00000000
orig_addr......... 00000000 orig_count..... 00000000
partial_stride... 00000000 first_issued.... 00000000
orig_bflags...... 00000000 pb_forw........ 0000 pb_back........ 0000
```
volgrp subcommand

Purpose
The volgrp subcommand displays volume group information. The volgrp structure addresses are registered in the devsw table in the DSDPTR field.

Syntax
volgrp [effectiveaddress]

Parameters
effectiveaddress Specifies the effective address of the volgrp structure to display. Use symbols, hexadecimal values or hexadecimal expressions to specify the address.

Aliases
No aliases.

Example
The following is an example of how to use the volgrp subcommand:

KDB(0)> devsw 0a

Slot address 0571E280
MAJOR: 00A
  open: 01B44DE4
  close: 01B44470
  read: 01B43C00
  write: 01B43C04
  ioctl: 01B42B18
  strategy: .hd_strategy
  tty: 00000000
  select: .nove
  config: 01B413A0
  print: .nove
  dump: .hd_dump
  mpx: .nove
  revoke: .nove
  dsdptr: 34D6C000
  selptr: 00000000
  opts: 0000000A DEV_DEFINED DEV_MPSAFE

KDB(0)> volgrp 34D6C000
VOLGRP...... 34D6C000
  vg_lock........... @ 34D6C000 vg_lock........... 00000000
  partshift........... 00000010
  open_count........... 00000009 flags........... 00000000
  lvols........... @ 34D6C02C
  pvolts........... @ 34D6C82C major_num........... 0000000A
  vg_id........... @ 0009FFFA0000000400000000F9E78590CE
  nextvgs........... 00000000 opn_pin........... @ 34D6CA2C
  von_pid........... 00000000 nxtactvg........... @ 34D6CA88
  ca_freevvw........... 00000000 ca_pvmmem........... 00000000
  ca_hld........... @ 34D6CA7C ca_pv_wrt........... @ 34D6CA88
  ca_inflct_cnt........... 00000000 ca_size........... 00000000
  ca_pvwbked........... 00000000 mwc_rec........... 00000000
  ca_part2........... 00000000 ca_lst........... 00000000
  ca_hash........... @ 34D6CAAC bcachwait........... FFFFFFFF
  ecachwait........... FFFFFFFF wait_cnt........... 00000000
  quorum_cnt........... 00000002 wheel_idx........... 00000000
  whl_seq_num........... 00000000 sa_act_lst........... 00000000
sa_hld_lst........... 00000000 vgsa_ptr............ 34D6E000
config_wait........... FFFFFFFF sa_lbuf............ 0 34D6CB10
sa_pbuf............ 0 34D6CB68
sa_intlock.......... 0 34D6CC0C sa_intlock........... 00000000
gv_intlock.......... 0 34D6CC10 vg_intlock........... 00000000
refresh_Q.......... 0 34D6CC14
gs_clvm........... 0 34D6CC20
oclvm........... 0 34D6CC24
cp_pwaitq.......... 0 34D6CACC
LVOL[000]........... 3004AF00
work_Q.......... 00000000 lv_status....... 00000000
lv_options...... 00000001 nparts........... 00000001
i_sched........ 00000000 nblocks........ 00200000
parts[0].......... 34D29A00 pvol0 34D90C00 dev 00170001 start 00000000
parts[1]........ 00000000
parts[2]........ 00000000
maxsize.......... 00000000 tot_rds.......... 00000000
complcnt........ 00000000 waitlist........ FFFFFFFF
stripe_exp....... 00000000 striping_width... 00000000
lv_intlock.@ 3004AF3C lv_intlock.... 00000000
LVOL[001]...........

...
**pvol subcommand**

**Purpose**
The `pvol` subcommand displays the physical volume data structure.

**Syntax**
`pvol [effectiveaddress]`

**Parameters**
*effectiveaddress* Specifies the effective address of the `pvol` structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `pvol` subcommand:

```
KDB(0)> pvol 34D6A000
PVOL........... 34D6A000
dev.............. 000C8010 xfcnt.............. 00000000
pvstate......... 00000029
pvnum............ FFFFD47C vg_num............. 00000000
fp............... 000A0003 flags.............. 00000000
num_bbdir_ent.... FFF8870 fst_usr_blk....... 0116D000
beg_relblk........ 00001000 next_relblk....... 00000000
max_relblk........ 00001000 defect_tbl........ 00000000
sa_area[0]....... @ 34D6A038
sa_area[1]....... @ 34D6A040 pv_pbuf.......... @ 34D6A048
oclvm............ @ 34D6A0F0
```
Ivol subcommand

Purpose
The ivol subcommand displays logical volume information.

Syntax
ivol [effectiveaddress]

Parameters
 effectiveaddress Specifies the effective address of the ivol structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Aliases
No aliases.

Example
The following is an example of how to use the ivol subcommand:

KDB(0)> ivol 3004AF00
LVOL............. 3004AF00
 work_Q.......... 00000000 lv_status....... 00000000
 lv_options..... 00000001 nparts.......... 00000001
 i_sched......... 00000000 nblocks......... 00200000
 parts[0].... 34D29A00 pvo10 34D90C00 dev 00170001 start 00000000
 parts[1]....... 00000000
 parts[2]....... 00000000
 maxsize........ 00000000 tot_rds......... 00000000
 complcnt........ 00000000 waitlist........ FFFFFF
 stripe_exp...... 00000000 striping_width.. 00000000
 ivol_intlock. @ 3004AF3C ivol_intlock.... 00000000
**scd subcommand**

**Purpose**

The `scd` subcommand displays the `scdisk_diskinfo` structure.

**Syntax**

`scd [slot | effectiveaddress]`

**Parameters**

*slot* Specifies the slot number of the scdisk entry to be displayed. To use this parameter, the scdisk list must have been previously loaded using the `scd` subcommand with no parameter. This value must be a decimal number.

*effectiveaddress* Specifies the effective address of an `scdisk_diskinfo` structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no argument is specified, the `scd` subcommand loads the slot numbers with addresses from the scdisk_list array. If the scdisk_list symbol cannot be located to load these values, the user is prompted for the address of the scdisk_list array. Obtain this address by locating the data address for the scdiskpin kernel extension and adding the offset to the scdisk_list array, which is obtained from a map, to that value.

A specific scdisk_list entry can be displayed by specifying either a slot number or the effective address of the entry. You can only use a slot number if the slots were previously loaded using the `scd` subcommand with no arguments.

**Aliases**

`scdisk`

**Example**

The following is an example of how to use the `scd` subcommand:

```
KDB(4)> lke 80 //print kernel extension information
ADDRESS FILE SIZE FLAGS MODULE NAME
80 05630900 01A57E60 0000979C 00000262 /etc/drivers/scdiskpin
1e_flags........ TEXT DATAINTEXT DATA DATAEXISTS
1e_fp............ 00000000
1e_loadcount.... 00000000
1e_usecount..... 00000001
1e_data/1e_tid.. 01A61320 <--- //this address plus the offset to
1e_data/1e_size.. 0000020C //the scdisk_list array (from a map)
1e_exports....... 0565E400 //are used to initialize the slots for
1e_lex........... 00000000 //the scd subcommand.
1e_defered...... 00000000
1e_filename..... 05630944
1e_ndepend...... 00000001
1e_maxdepend..... 00000001
1e_de............ 00000000
KDB(4)> d 01A61320 100 //print data
01A61320: 0000 0008 0000 0006 FFF FFF 0562 7C00 ...............b|.
01A61330: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
01A61340: 01A6 080C 01A6 0808 01A6 0804 01A6 0800 ................
01A61350: 01A6 08CC 01A6 08C8 01A6 08C4 01A6 08C0 ................
01A61360: 01A6 0920 01A6 0960 01A6 09A0 01A6 09ED ............
01A61370: 01A6 0A20 01A6 0A60 01A6 0A00 01A6 0AEO ............
01A61380: 01A6 0B20 01A6 0B60 01A6 0BA0 01A6 0BED ............
01A61390: 01A6 0C20 01A6 0C60 01A6 0CA0 01A6 0C00 ............
01A613A0: 7363 696E 666F 0000 6366 676C 6973 7400 scinfo.cfglist.
```
Unable to find <scdisk_list>
Enter the scdisk_list address (in hex): 01A61418
Scsi pointer [01A61418]
slot 0...........05674000
slot 1...........05675000
slot 2...........0566C000
slot 3...........0566D000
slot 4...........0566E000
slot 5...........0566F000
slot 6...........05670000
slot 7...........05671000
slot 8...........05672000
slot 9...........05673000
slot 10..........0C40D000
slot 11...........00000000
slot 12...........00000000
slot 13...........00000000
slot 14...........00000000
slot 15...........00000000

KDB(4)> scd //print scsi disk table
Scdisk info [05674000]
next......................00000000 next_open..................00000000
devno.....................00120000 adapter_devno...........00100000
watchdog_timer.watch.0....05674010 watchdog_timer.pointer...05674000
scsi_id...................00000000 lun_id....................00000000
reset_count...............00000000 dk_cmd_q_head............00000000
dk_cmd_q_tail...............00000000 ioctl_cmd0...............05674034
cmd_pool...................05628400 pool_index..............00000000
open_event..................FFFFFFFH checked_cmd..........00000000
writev_cmd0...............056740FC reassign_err_cmd........00000000
reset_cmd0...............056740AC regnsns_cmd0...........056741AC
writev_cmd1...............0567425C q_recoq_cmd0...........0567430C
dk_bp_queue0...............0567451C mode......................00000001
disk_intrpt...............00000000 raw_io_intrpt...........00000000
ioctl_chg_mode_flg........00000000 m_sense_status........00000000
opened....................00000001 cmd_pending............00000000
eerrno.....................00000000 retain_reservation........00000000
q_type....................00000000 q_err_value............00000001
clr_q_on_error............00000000 buffer_ratio............00000000
ioctl_tag_q...............00000000 q_status...............00000000
q_clr......................00000000 timer_status...........00000000
restart_unit.............00000000 retry_flag.............00000000
(4)> more ("^C to quit")? /continue

KDB(4)>
rw_timeout................0000001E fmt_timeout...............00000000
queue_depth...............00000001D0 cmds_out..................00000000
raw_io_cmd................00000000 currbuf...............10A0546E0
low.......................10A14E3C0 block_size........200
cfg_block_size............200 last Ses_pvd_lba...........00000000
max_request...............00040000 max_coalesce...........00010000
lock......................FFFFFFFF fp........................00414348
more(^C to quit) ? //continue
error_rec@................05674598 stats@..................05674648
stats@....................0567465C disc_info@..............05674660
mode_list_head@...........05674A63 ioctl_buf@..............05674A64
ioctl_req_sense@..........05674C8C capacity@.................05674CA4
def_list@..................05674CAC def_list_header@...........05674B63
dd@.......................05674B6C df@.......................05674BB4
cd@.......................05674BFC spin_lock@...............05674CF8
spin_lock.................E80039A0 pmh@......................05674CFC
pm_pending................00000000 pm_device_id............00100000
pm_event..................FFFFFFFF pm_timer@.................05674D4C
KDB(4)> file 00414348 //print file (fp)

COUNT OFFSET DATA TYPE FLAGS
18 file+000330 1 0000000000000000 0BC4A950 GNODE WRITE

f_flag.................00000002 f_count...........00000001
f_msgcount............00000003 f_type..............003
fdata...............0BC4A950 f_offset......000000000000000000
fdir_off..............00000000 f_cred...........00000000
flock...............00414368 f_lock..........00414368
f_offset_lock........0041436C f_offset_lock..0041436C
f_vinfo..............00000000 f_ops..........001F3CD0
gno_fops+000000
GNODE.............0BC4A950

gn_seg..............007FFFFF gn_mwrcnt....00000000 gn_mrdcnt....00000000
gn_rdcnt............00000000 gn_wrcnt......00000000 gn_excnt......00000000
gn_rshcnt............00000000 gn_ops.........00000000 gn_vnode......00000000
gn_reclk............00000000 gn_rdev......00000000
gn_chan..............00000000 gn_filocks...00000000 gn_data......0BC4A940
gn_type............BLK

KDB(4)> buf 0A0546E0 //print current buffer (currenbuf)
DEV VNODE BLKNO FLAGS
 0 0A0546E0 01200000 00000000 00070A5B READ SPLIT MPSAFE MPSAFE_INITIAL

forw 00000000 back 00000000 av_forw 0A05DC0 av_back 0A14E3C0
blkno 00070A5B addr 00626000 bcount 00001000 resid 00000000
error 00000000 work 00000000 options 00000000 event FFFFFFFF
iodone: 019057D4
start.tv_sec......00000000 start.tv_nsec.....00000000
xmemd.aspace_id 00000000 xmemd.xm_flag 00000000 xmemd.xm_version 00000000
xmemd.subspace_id 00800802 xmemd.subspace_id2 00000000 xmemd.uaddr 00000000

Chapter 28. Display storage subsystem information subcommands  295
Chapter 29. Display memory allocation information subcommands

The subcommands in this category display memory allocation information. These subcommands include the following:

- `heap`
- `xm`
- `kmbucket`
- `kmstats`
heap subcommand

Purpose
The heap subcommand displays information about heaps.

Syntax
heap [\[-l|f|d|a\] [\(<address> <SRAD number>\)]

Parameters
-\(l\) Displays information about the heap, including the complete free page list, deferred free page list (MODS-enabled only), and allocated pages.
-\(f\) Displays only the free page list.
-\(d\) Displays only the deferred free page list.
-\(a\) Displays only the allocated pages.
\(<address>\) Specifies the effective address of the heap. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.
\(<SRAD number>\) Specifies that the heap to display should be the default kernel heap corresponding to the given system resource allocation domain (SRAD).

Aliases
hp

Example
The following is an example of how to use the heap subcommand:

```
KDB(0)> heap
Pinned heap 02DACFA8
sanity........ 4E554D41 alt........... 00000001
heapaddr[00]... F1000C00000000000 [01].. 0
heapaddr[02]... 0 [03].. 0
heapaddr[04]... 0 [05].. 0
heapaddr[06]... 0 [07].. 0
heapaddr[08]... 0 [09].. 0
heapaddr[10]... 0 [11].. 0
heapaddr[12]... 0 [13].. 0
heapaddr[14]... 0 [15].. 0
baseaddr[00]... F1000C000FF0A000 [01].. 0
baseaddr[02]... 0 [03].. 0
baseaddr[04]... 0 [05].. 0
baseaddr[06]... 0 [07].. 0
baseaddr[08]... 0 [09].. 0
baseaddr[10]... 0 [11].. 0
baseaddr[12]... 0 [13].. 0
baseaddr[14]... 0 [15].. 0
numpds[00].... FF00F6 [01].. 0
numpds[02].... 0 [03].. 0
numpds[04].... 0 [05].. 0
numpds[06].... 0 [07].. 0
numpds[08].... 0 [09].. 0
numpds[10].... 0 [11].. 0
numpds[12].... 0 [13].. 0
numpds[14].... 0 [15].. 0
12pd_size[00].. C [01].. 0
12pd_size[02].. 0 [03].. 0
12pd_size[04].. 0 [05].. 0
12pd_size[06].. 0 [07].. 0
12pd_size[08].. 0 [09].. 0
12pd_size[10].. 0 [11].. 0
```
l2pd_size[12].. 0 [13].. 0
l2pd_size[14].. 0 [15].. 0
Kernel heap 020AD150
sanity........ 4E554D41 alt............ 00000000
heapaddr[00].. F1000C0000000110 [01].. 0
heapaddr[02].. 0 [03].. 0
heapaddr[04].. 0 [05].. 0
heapaddr[06].. 0 [07].. 0
heapaddr[08].. 0 [09].. 0
heapaddr[10].. 0 [11].. 0
heapaddr[12].. 0 [13].. 0
heapaddr[14].. 0 [15].. 0
baseaddr[00].. F1000C0000FF0A000 [01].. 0
baseaddr[02].. 0 [03].. 0
baseaddr[04].. 0 [06].. 0
baseaddr[06].. 0 [07].. 0
baseaddr[08].. 0 [09].. 0
baseaddr[10].. 0 [11].. 0
baseaddr[12].. 0 [13].. 0
baseaddr[14].. 0 [15].. 0
numpds[00].... FF00F6 [01].. 0
numpds[02].... 0 [03].. 0
numpds[04].... 0 [05].. 0
numpds[06].... 0 [07].. 0
numpds[08].... 0 [09].. 0
numpds[10].... 0 [11].. 0
numpds[12].... 0 [13].. 0
numpds[14].... 0 [15].. 0
l2pd_size[00].. C [01].. 0
l2pd_size[02].. 0 [03].. 0
l2pd_size[04].. 0 [05].. 0
l2pd_size[06].. 0 [07].. 0
l2pd_size[08].. 0 [09].. 0
l2pd_size[10].. 0 [11].. 0
l2pd_size[12].. 0 [13].. 0
l2pd_size[14].. 0 [15].. 0
KDB(0)> heap 0
Heap for node #00
Heap F1000C0000000110
sanity........ 48454150 base... 000000000FF09EF0
lock@.. F1000C0000000120 lock... 0000000000000000
alt............ 00000000 numpds........ 00FF00F6
amount. 000000000B106FF0 pinflag........ 00000000
vmrelflag...... 00000000 newheap........ 00000000
protect........ 00000000 limit.......... 00000000
heap64.......... 00000000 rhash.. 0000000002E56048
pagtot........ 0001223C pagused....... 00000000
frtot[00].. 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
frtot[04].. 0000000001F8 [05].. 0000000001F8 [06].. 0000000001F8 [07].. 0000000001F8
frused[00].. 0000000001F8 [01].. 0000000001F8 [02].. 0000000001F8 [03].. 0000000001F8
frused[04].. 0000000001F8 [05].. 0000000001F8 [06].. 0000000001F8 [07].. 0000000001F8
frused[08].. 0000000001F8 [09].. 0000000001F8 [10].. 0000000001F8 [11].. 0000000001F8
fr[00].... 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
fr[04].... 0000000001F8 [05].. 0000000001F8 [06].. 0000000001F8 [07].. 0000000001F8
fr[08].... 0000000001F8 [09].. 0000000001F8 [10].. 0000000001F8 [11].. 0000000001F8
Heap: per-cpu free lists
Heap F1000C0000000110
sanity........ 48454150 base... 000000000FF09EF0
lock@.. F1000C0000000120 lock... 0000000000000000
alt............ 00000000 numpds........ 00FF00F6
amount. 000000000B106FF0 pinflag........ 00000000
vmrelflag...... 00000000 newheap........ 00000000
protect........ 00000000 limit.......... 00000000
heap64.......... 00000000 rhash.. 0000000002E56048
pagtot........ 0001223C pagused....... 00000000
frtot[00].. 0000000001F8 [01].. 0000000001F8 [02].. 0000000001F8 [03].. 0000000001F8
frtot[04].. 0000000001F8 [05].. 0000000001F8 [06].. 0000000001F8 [07].. 0000000001F8
frtot[08].. 0000000001F8 [09].. 0000000001F8 [10].. 0000000001F8 [11].. 0000000001F8
fr[00].... 00FFFFFF [01].. 00FFFFFF [02].. 00FFFFFF [03].. 00FFFFFF
fr[04].... 0000000001F8 [05].. 0000000001F8 [06].. 0000000001F8 [07].. 0000000001F8
fr[08].... 0000000001F8 [09].. 0000000001F8 [10].. 0000000001F8 [11].. 0000000001F8
Chapter 29. Display memory allocation information subcommands 299
Heap: per-cpu free lists
addr........ F1000C00FF0A000 overflow........ 00000000
ov_min........ 0000000000000000 ov_max........ 0000000000000000
maxpds........ 00FF00F6 peakd........ 00000000
l2pg_size....... 00000000C l2pd_size....... 00000000C
limit_coutout.... 00000000 newseg_coutout..... 00000000
pages_offset.... 00008BA0 pages_serval........ 00000000
lock_offset..... 00000400 locks_serval........ 00000000
heap_locks...... @ F1000C0000000400 pages........... @ F1000C0000008BA0
heap_lists...... @ F1000C000000A80
Heap anchor
... F1000C0000008B90 pageno FFFFFFFF type.. 00 allocpage offset... 0158C9
  The largest free range in the heap is 1623661 pages and starts
  at page 158C9.
KDB(0)> heap -l 0
Heap for node #0
Heap F1000C0000000110
sanity......... 48454150 base... 0000000000FF09EF0
lock0... F1000C0000000120 lock... 0000000000000000
alt............ 00000000 numpds........ 00FF00F6
amount. 0000000000B106FF0 pinflag........ 00000000
vmreflag....... 00000000 newheap........ 00000000
protect........ 00000000 limit.......... 00000000
heap64........ 00000000 rhash.. 000000002E56048
pagetot........ 0000232C pagused........ 000AFAA
frotd[04].... 00000000 [01]... 00000000 [02]... 00000000 [03]... 00000000
frused[00].... 00000000 [01]... 00000000 [02]... 00000000 [03]... 00000000
frused[04].... 00000000 [05]... 000005F6 [06]... 00000073 [07]... 0000001E0
frused[08].... 00000000A3 [09]... 00000026 [10]... 0000000B [11]... 000000012
fr[00]...... 00FF00FF [01]... 00FF00FF [02]... 00FF00FF [03]... 00FF00FF
fr[04]...... 000000E0 [05]... 0000F18B [06]... 00158C6 [07]... 00158B6
fr[08]...... 0015242 [09]... 000F216 [10]... 00158BF [11]... 000F3D3
Heap: per-cpu free lists
addr........ F1000C00FF0A000 overflow........ 00000000
ov_min........ 0000000000000000 ov_max........ 0000000000000000
maxpds........ 00FF00F6 peakd........ 00000000
l2pg_size....... 00000000C l2pd_size....... 00000000C
limit_coutout.... 00000000 newseg_coutout..... 00000000

300  KDB Kernel debugger and kdb command
Chapter 29. Display memory allocation information subcommands  

Heap anchor
... F1000C0000008B90 pageno FFFFFFFF type.. 00 allocpage offset... 0158C9
Heap Free list
... F1000C00000161830 pageno 000158C9 type.. 03 freerange offset... NO_PAGE
... F1000C00000161840 pageno 000158CA type.. 04 freesize size...... FDAB2D
... F1000C0000FF09A0 pageno 00FF00F5 type.. 05 freerangeend offset... 0158C9
Heap deferred free anchor
... F1000C0000008B80 pageno FFFFFFFF type.. 00 allocpage offset... 0158C7
Heap Deferred Free list
... F1000C00000161870 pageno 000158C7 type.. 01 allocrange offset... 0158C0
... F1000C00000161820 pageno 000158CB type.. 06 allocsize size...... 000002
... F1000C000001617A0 pageno 000158C0 type.. 01 allocrange offset... 0158B7

Heap Alloc list
... F1000C0000008B0 pageno 00000000 type.. 00 allocpage offset... 0158C0
... F1000C0000008B0 pageno 00000001 type.. 00 allocpage offset... 0158C0
... F1000C0000008B0 pageno 00000002 type.. 00 allocpage offset... 0158C0

KOB(0)> heap -f 0
Heap for node #00
Heap F1000C0000000110
Heap: per-cpu free lists
Heap F1000C0000000000
Heap: per-cpu free lists
addr...... F1000C000FF0A000 overflow........... 00000000
ov_min...... 0000000000000000 ov_max...... 0000000000000000
maxpds........... 00FF0F6 peakpd............. 00000000
12pg_size....... 0000000C 12pd_size............ 0000000C
limit_callout...... 00000000 newseg_callout...... 00000000
pagesoffset...... 0000BBA0 pagessrval........... 00000000
lockoffset...... 00000400 locks_srchval...... 00000000
heap_locks.....  @ F1000C0000000400 pagesrval.....  @ F1000C000008B0
heap_lists..... @ F1000C0000000A0
Heap anchor
... F1000C0000008B90 pageno FFFFFFFF type.. 00 allocpage offset... 0158C9
Heap Free list
... F1000C00000161830 pageno 000158C9 type.. 03 freerange offset... NO_PAGE
... F1000C00000161840 pageno 000158CA type.. 04 freesize size...... FDAB2D
... F1000C0000FF09A0 pageno 00FF00F5 type.. 05 freerangeend offset... 0158C9
KOB(0)> heap -d 0
Heap for node #00
Heap F1000C0000000110
Heap: per-cpu free lists
Heap F1000C0000000000
Heap: per-cpu free lists
addr...... F1000C000FF0A000 overflow........... 00000000
ov_min...... 0000000000000000 ov_max...... 0000000000000000
maxpds........... 00FF0F6 peakpd............. 00000000
12pg_size....... 0000000C 12pd_size............ 0000000C
limit_callout...... 00000000 newseg_callout...... 00000000
pagesoffset...... 0000BBA0 pagessrval........... 00000000
lockoffset...... 00000400 locks_srchval...... 00000000
heap_locks.....  @ F1000C0000000400 pagesrval.....  @ F1000C000008B0
heap_lists..... @ F1000C0000000A0
Heap anchor
... F1000C0000008B90 pageno FFFFFFFF type.. 00 allocpage offset... 0158C9
Heap Free list
... F1000C00000161830 pageno 000158C9 type.. 03 freerange offset... NO_PAGE
... F1000C00000161840 pageno 000158CA type.. 04 freesize size...... FDAB2D
... F1000C0000FF09A0 pageno 00FF00F5 type.. 05 freerangeend offset... 0158C9

The largest free range in the heap is 16623661 pages and starts
at page 158C9.

Heap deferred free anchor
... F1000C0000008B80 pageno FFFFFFFF type.. 00 allocpage offset... 0158C7
Heap Deferred Free list
... F1000C00000161870 pageno 000158C7 type.. 01 allocrange offset... 0158C0
... F1000C00000161820 pageno 000158CB type.. 06 allocsize size...... 000002
... F1000C00001617A0 pageno 000158C0 type.. 01 allocrange offset... 0158B7

KDB(0)> heap -d 0
Heap for node #00
Heap: per-cpu free lists
Heap: per-cpu free lists
addr... F1000C0000FF0A000 overflow......... 00000000
ov_min... 0000000000000000 ov_max.... 0000000000000000
maxpds........ 00F00F6 peakpd......... 00000000
l2pg_size........ 0000000C l2pd_size......... 0000000C
limit_callout..... 00000000 newseg_callout.... 00000000
pagesoffset...... 0000B8A0 pages_srsval..... 00000000
lockoffset........ 00004000 locks_srsval..... 00000000
heap_locks ..... @ F1000C0000000400 pages......... @ F1000C000000B8A0
heap_lists ..... @ F1000C0000000A80
Heap anchor
... F1000C0000000B90 pageno FFFFFFF type.. 00 allocpage offset... 0158C9
  The largest free range in the heap is 16623661 pages and starts
  at page 158C9.
Heap Alloc list
... F1000C0000000B8A0 pageno 00000000 type.. 00 allocpage offset... NO_PAGE
... F1000C0000000B8B0 pageno 00000001 type.. 00 allocpage offset... NO_PAGE
... F1000C0000000B8C0 pageno 00000002 type.. 00 allocpage offset... NO_PAGE

KDB(0)> heap @kernel_heap0
Heap 03AF80E0
sanity........ 48454150 base... 0000000000039F20
lock0... 0000000003AF80F0 lock... 0000000000000000
alt........... 00000000 numpages........ 000C4CE
amount. 00000000005E66C0 pinflag........ 00000000
vmaeflag.... 00000000 newheap........ 00000000
protect....... 00000000 limit........... 00000000
heap64....... 00000000 rhash... 0000000000000000
pagetot.... 00000000 pagused........ 00000000
...
xmalloc subcommand

Purpose
The xmalloc subcommand displays memory allocation information, finds the memory location of any heap record using the page index or finds the heap record using the allocated memory location.

Syntax
xmalloc [-s {effectiveaddress}] [-h {effectiveaddress}] [[-l] [-f] [-a] [-l] [-p page] [-d {effectiveaddress}] [-v] [[-q] [-u {size}] [-S {effectiveaddress}] [-H heap_addr]]

Parameters
- **-s** Displays allocation records matching the value of the effectiveaddress parameter. If the effectiveaddress parameter is not specified, the value of the Debug_addr symbol is used.
- **-h** Displays free list records matching effectiveaddress. If effectiveaddress is not specified, the value of the Debug_addr symbol is used.
- **-l** Enables verbose output. Applicable only with the -f, -a, and -p flags.
- **-f** Displays records on the free list, from the first freed record to the last freed record.
- **-a** Displays allocation records.
- **-p page** Displays page information for the specified page. The page number is a hexadecimal value.
- **-d** Displays the allocation record hash chain associated with the record hash value for the effectiveaddress parameter. If the effectiveaddress parameter is not specified, the value of the Debug_addr symbol is used.
- **-v** Verifies allocation trailers for allocated records and verifies free fill patterns for free records.
- **-q** Indicates that allocations should not be separated into size groups.
- **-u** Displays heap statistics.
- **size** Specifies the largest size allocation reported.
- **-S** Displays heap locks and per-processor lists.
  
  Note: The per-processor lists are only used for the kernel heaps.

**effectiveaddress** Specifies the effective address for which information is to be displayed. Use symbols, hexadecimal values, or hexadecimal expressions to specify the effective address.

**-H heap_addr** Specifies the effective address of the heap for which information is displayed. If the -H parameter is not specified, information is displayed for the kernel heap. The -H parameter can be supplied with other xmalloc parameters. Use symbols, hexadecimal values, or hexadecimal expressions to specify the effective address.

Other than the -u parameter, these parameters require that the Memory Overlay Detection System (MODS) is active. If parameters require a memory address and no value is specified, the value of the Debug_addr symbol is used. If a system crash is caused by detection of a problem within MODS, this value is updated by MODS. The default heap reported on is the kernel heap.

Aliases
xm

Example
The following is an example of how to use the xm alias of the xmalloc subcommand:

(0)> stat
RS6K_SMP_MCA POWER_PC POWER_604 machine with 8 processor(s)
........... SYSTEM STATUS
sysname... AIX nodename.. jumbo32
release... 3 version... 4
machine... 00920312A0 nid....... 920312A0
time of crash: Fri Jul 11 08:07:01 1997
age of system: 1 day, 20 hr., 31 min., 17 sec.
PANIC STRING
Memdbg: *w == pat

(0)> xm -s //Display debug xmalloc status
Debug kernel error message: The xmfree service has found data written beyond the end of the memory buffer that is being freed.
Address at fault was 0x09410200

(0)> xm -h 0x09410200 //Display debug xmalloc records associated with addr

0364120: addr......... 09410200 req_size..... 128 freed unpinned
0364120: pid......... 00043158 comm.......... bcross
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00235C04(.dlistadd+000040) 00234F04(.setbitmaps+0001BC)
00235520(.newblk+00006C) 00236894(.finicom+0001A4)

0664120: addr......... 09410200 req_size..... 128 freed unpinned
0664120: pid......... 0007DCAC comm.......... bcross
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00235C04(.dlistadd+000040) 00234F04(.setbitmaps+0001BC)
00235520(.newblk+00006C) 00236894(.finicom+000030)

07A3750: addr......... 09410200 req_size..... 128 freed unpinned
07A3750: pid......... 000010BA comm.......... syncd
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00235C04(.dlistadd+000040) 00234F04(.setbitmaps+0001BC)
00235520(.newblk+00006C) 00236894(.finicom+000030)

0B52B30: addr......... 09410200 req_size..... 128 freed unpinned
0B52B30: pid......... 00058702 comm.......... bcross
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00235C04(.dlistadd+000040) 00234F04(.setbitmaps+0001BC)
00236510(.logdfree+0000E4) 00236720(.finicom+000030)

07A33840: addr......... 09410200 req_size..... 133 freed unpinned
07A33840: pid......... 000010BA comm.......... ksh
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160) 00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074) 002ABF04(.ld_execload+000064)

07B96480: addr......... 09410200 req_size..... 133 freed unpinned
07B96480: pid......... 0005C2E0 comm.......... ksh
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160) 00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074) 002ABF04(.ld_execload+000075)

07A31420: addr......... 09410200 req_size..... 135 freed unpinned
07A31420: pid......... 0007161A comm.......... ksh
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160) 00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074) 002ABF04(.ld_execload+000075)

07A3630: addr......... 09410200 req_size..... 125 freed unpinned
07A3630: pid......... 0001121E comm.......... ksh
Trace during xmalloc() Trace during xmfree()
002329E4(.xmalloc+0000A8) 002328F0(.xmfree+0000FC)
00271F28(.ld_pathopen+000160) 00271D24(.ld_pathclear+00008C)
0027FB6C(.ld_getlib+000074) 002ABF04(.ld_execload+000075)

07A3D240: addr......... 09410200 req_size..... 133 freed unpinned
07A3D240: pid......... 0000654C comm.......... ksh
Trace during xmalloc()
002329E4(.xmalloc+0000A8)
00271F28(.ld_pathopen+000160)
0027F66C(.ld_getlib+000074)

Trace during xmfree()
002328F0(.xmfree+0000FC)
00271D24(.ld_pathclear+00008C)
002ABF04(.ld_execload+00075C)

(0)> heap...
Heap Alloc list
... 0FFC41B0 pageno 00000007 pages.type.. 01 allocrange offset... NO_PAGE
... 0FFC89AC pageno 00001E06 pages.type.. 06 allocsize size..... 0001E00
... 0FFC89B0 pageno 00001E07 pages.type.. 01 allocrange offset... NO_PAGE
... 0FFC89B4 pageno 00001E08 pages.type.. 06 allocsize size..... 0001E00
... 0FFD31AC pageno 00003C06 pages.type.. 07 allocrangeend offset... 00000007
... 0FFD31B4 pageno 00003C08 pages.type.. 01 allocrange offset... 00003C42
... 0FFD31B8 pageno 00003C09 pages.type.. 06 allocsize size..... 00000002
... 0FFD31C4 pageno 00003C0C pages.type.. 01 allocrange offset... NO_PAGE
... 0FFD31C8 pageno 00003C0D pages.type.. 06 allocsize size..... 00000009
... 0FFD31E4 pageno 00003C14 pages.type.. 07 allocrangeend offset... 00003C0C

(0)> x -l -p 00001E07 //how to find memory address of heap index 00001E07

type.................... 1 (P_allocrange)
page_addr.............. 02F82000 pinned................. 0
size.................... 00000000 offset................ 00000000
page_descriptor_address.. 0FFCB9B0

(0)> x -l -p 00003C08 //how to find page index in kernel heap of 02F82000
P_allocrange (range of 2 or more allocated full pages)

page........... 00001E07 start........ 02F82000 page_cnt..... 00001E00
allocated_size. 01E000000 pinned........ unknown

(0)> x -l -p 04D83000 //how to find memory address of heap index 00003C0C

type.................... 1 (P_allocrange)
page_addr.............. 04D830000 pinned................. 0
size.................... 00000000 offset................ 00003C42
page_descriptor_address.. 0FFD31B4

(0)> x -l -p 04D83000 //how to find page index in kernel heap of 04D83000
P_allocrange (range of 2 or more allocated full pages)

page........... 00003C08 start........ 04D83000 page_cnt..... 00000002
allocated_size. 00002000 pinned........ unknown
**kmbucket subcommand**

**Purpose**
The kmbucket subcommand prints kernel memory allocator buckets.

**Syntax**

```
kmbucket [-l] [-c cpu] [-i index] [effectiveaddress]
```

```
kmbucket -k effectiveaddress
```

```
kmbucket -s
```

**Parameters**

- `-l` Displays the bucket free list.
- `-c cpu` Displays only buckets for the specified processor. Specify the cpu parameter as a decimal value.
- `-i index` Displays only the bucket for the specified index. The index is specified as a decimal value.
- `effectiveaddress` Specifies the effective address of the kmembucket structure to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.
- `-k` Displays the kmemusage structure associated with the `effectiveaddress`.
- `-s` Displays the netkmem structure.

If no arguments are specified, information is displayed for all allocator buckets for each processor.

**Aliases**

bucket

**Example**
The following is an example of how to use the kmbucket subcommand:

```
KDB(0)> kmbucket -c 0 -i 11
displaying kmembucket for cpu 0 offset 11 size 0x000000800

address.................F10006000BD8BD48 b_next...(x)...........F100061002AD1000
b_calls...(x)...........000000000001405 b_total...(x)...........0000000000080A
b_totalfree...(x).......0000000000000006 b_elmpercl...(x)...........00000000000002
b_highwat...(x).........00000000000007AD b_couldfree (sic).(x)...........00000000000000
b_failed...(x)..........0000000000000000 b_delayed..............0000000000000000
lock.................... 0 F10006000BD8BD90 lock...(x).............0000000000000000
delta...............FFFFFFFFFFFFD800
KDB(0)> kmbucket F10006000BD8BD48 //address field from above

displaying kmembucket for cpu 0 offset 11 size 0x000000800

address.................F10006000BD8BD48 b_next...(x)...........F100061002ACB000
b_calls...(x)...........000000000001407 b_total...(x)...........0000000000080A
b_totalfree...(x).......0000000000000005 b_elmpercl...(x)...........00000000000002
b_highwat...(x).........00000000000007AD b_couldfree (sic).(x)...........00000000000000
b_failed...(x)..........0000000000000000 b_delayed..............0000000000000000
lock.................... 0 F10006000BD8BD90 lock...(x).............0000000000000000
delta...............FFFFFFFFFFFFE000

Bucket free list....
1 next........F100061002ACB000 prev...00000000,
    kmemusage...F10006000BE08308 [000B 0002 00000000]
2 next........F100061002AE0800 prev...F100061002ACB000,
    kmemusage...F10006000BE08500 [000B 0001 00000000]
```

306 KDB Kernel debugger and kdb command
This address belongs to the following kmemusage structure:

- **ku_indx**: `0000000B`
- **free/page cnt**: `00000002`
- **ku_cpu**: `00000000`
**kmstats subcommand**

**Purpose**
The *kmstats* subcommand prints kernel allocator memory statistics.

**Syntax**
```
kmstats [effectiveaddress]
```

**Parameters**
- `effectiveaddress`: Specifies the effective address of the kernel allocator memory statistics entry to display. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no address is specified, all of the kernel allocator memory statistics are displayed. If an address is entered, only the specified statistics entry is displayed.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the *kmstats* subcommand:
```
KDB(0)> kmstats //print allocator statistics

displaying kmemstats for offset 0 free
address..................0025C120
inuse...(x)...............00000000
calls...(x)...............00000000
memuse...(x)............00000000
limit blocks...(x).....00000000
map blocks...(x)........00000000
maxused...(x)...........00000000
limit...(x).............02666680
failed...(x)............00000000
lock...(x)...............00000000

displaying kmemstats for offset 1 mbuf
address..................0025C144
inuse...(x)...............0000000D
calls...(x)...............002C4E54
memuse...(x)............0000D000
limit blocks...(x).....00000000
map blocks...(x)........00000000
maxused...(x)...........001D700
limit...(x).............02666680
(0)> more (^C to quit) ? //continue
failed...(x)............00000000
lock...(x)...............00000000

displaying kmemstats for offset 2 mcluster
address..................0025C168
inuse...(x)...............00000002
calls...(x)...............0023D04E
memuse...(x)............00009900
limit blocks...(x).....00000000
map blocks...(x)........00000000
maxused...(x)...........00079C00
limit...(x).............02666680
failed...(x)............00000000
```
lock..(x)...............0000000

...

displaying kmemstats for offset 48 kalloc
address.....................0025C7E0
inuse..(x)..................0000000
calls..(x)..................0000000
memuse..(x)................00000000
limit blocks..(x)..........00000000
map blocks..(x)............00000000
maxused..(x)................00000000
limit..(x)..................02666680
failed..(x)..................00000000
lock..(x)..................00000000

displaying kmemstats for offset 49 temp
address.....................0025C804
inuse..(x)..................000000007
calls..(x)..................000000007
memuse..(x)................00003500
(0)> more (^C to quit) ? //continue
limit blocks..(x)...........00000000
map blocks..(x)............00000000
maxused..(x)................00003500
limit..(x)..................02666680
failed..(x)..................00000000
lock..(x)..................00000000
KDB(0)>

Chapter 29. Display memory allocation information subcommands 309
Chapter 30. Display general and Journal File System (JFS) information subcommands

The subcommands in this category can be used to display general file system information, and information specific to the JFS filesystem. These subcommands include the following:

- “dnlc subcommand” on page 312
- “hdnlc subcommand” on page 314
- “kvn subcommand” on page 316
- “buffer subcommand” on page 317
- “hbuffer subcommand” on page 319
- “tbuffer subcommand” on page 320
- “gnode subcommand” on page 321
- “gfs subcommand” on page 322
- “file subcommand” on page 323
- “inode subcommand” on page 325
- “hinode subcommand” on page 328
- “icache subcommand” on page 329
- “vnc subcommand” on page 331
- “hvnc subcommand” on page 333
- “vnode subcommand” on page 335
- “vfs subcommand” on page 336
- “specnode subcommand” on page 338
- “devnode subcommand” on page 340
- “fifonode subcommand” on page 342
- “hnnode subcommand” on page 344
- “jfsnode subcommand” on page 345
- “kfset subcommand” on page 347
**dnlc subcommand**

**Purpose**
The `dnlc` subcommand displays information about the filesystem directory name lookup cache.

**Syntax**
```
dnlc [slot | effectiveaddress]
```

**Parameters**
- `slot` Specifies the decimal identifier of a specific cache slot.
- `effectiveaddress` Specifies the effective address of the entry. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

The `dnlc` subcommand is used to display information about the directory name cache.

When no parameters are provided, a summary of the entire directory name lookup cache is displayed.

**Aliases**
`ncache`

**Example**
The following is an example of how to use the `dnlc` subcommand:

```
KDB(0)> dnlc

DP         NP  NAME
1  KERN_heap+59B9000 F10000F0049FB848 F10000F004ED3D78 __vg10
2  KERN_heap+59B9060 F10000F00513FD78 F10000F00510A078 CuAt.vc
3  KERN_heap+59B90C0 F10000F0049FB848 F10000F004ED3D78 __pv16.0
4  KERN_heap+59B9120 F10000F0049FB848 F10000F00510A078 hd6
5  KERN_heap+59B9180 F10000F0049FB848 0000000000000000 __pv16.0
6  KERN_heap+59B91E0 F10000F0049FB848 F10000F004FD6ED78 __pv16.0
7  KERN_heap+59B9240 F10000F00557C918 F10000F005883918 libcrypt.a
8  KERN_heap+59B92A0 F10000F0049FB848 0000000000000000 __pv16.0
9  KERN_heap+59B9300 F10000F004B31B48 F10000F004B31B48 etc
10 KERN_heap+59B9360 F10000F005009D78 F10000F0050A0D78 CuAt
11 KERN_heap+59B93C0 F10000F004963D98 F10000F0051E1218 diagrpt23.dat
12 KERN_heap+59B9420 F10000F004B31B48 F10000F004B31B48 dev
13 KERN_heap+59B9480 F10000F005883918 F10000F004D02D78 vg
14 KERN_heap+59B94E0 F10000F004B31B48 F10000F00509D78 objrepos
<snip>
```

KDB(0)> dnlc 14 //slot

```
DP         NP  NAME
14 KERN_heap+59B94E0 F10000F004B31B48 F10000F00509D78 objrepos
```

vfsp...... F10000F0065F2470 forw...... F10000F0059B9300 back...... F10000F0059B9060
dp...... F10000F004B31B48 did....... 0000000A
np...... F10000F00509D78 nid....... 0000000F nidp...... F10000F00509D78
namelen.. 00000008
KDB(0)> dnlc F10000F0059B94E0 //eaddr

```
DP         NP  NAME
14 KERN_heap+59B94E0 F10000F004B31B48 F10000F00509D78 objrepos
```

vfsp...... F10000F0065F2470 forw...... F10000F0059B9300 back...... F10000F0059B9060
dp....... F10000F004B31B48 did...... 00000000
np....... F10000F005009D78 nid...... 000000FF nidp..... F10000F005009E38
namelen.. 00000000
KDB(0)> dnlc nlc_cache  //symbol
      DP     NP NAME
2146583247 nlc_cache+000000 000000000000000020000 0000000000000000D2198
vfsp..... 0000000000000020000 forw..... F10000F0059B90000 back..... 000000000020000
dp....... 00000000000020000 did...... 00000000
np....... 000000000022198 nid...... FFFFFFFF nidp..... 0000000000D73F60
namelen.. FFFFFFFF
KDB(0)>

Chapter 30. Display general and Journal File System (JFS) information subcommands
hdnlc subcommand

Purpose
The hdnlc subcommand displays information about the file system hash list for the directory name cache.

Syntax
hdnlc [slot | effectiveaddress]

Parameters
slot Specifies the decimal identifier of a specific hash bucket.
effectiveaddress Specifies the effective address of the entry. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the address.

The hdnlc command is used to display information about the dnlc hash table. When no parameters are provided, a summary of the entire hash list is displayed.

Aliases
hncache

Example
The following is an example of how to use the hdnlc subcommand:

KDB(0)> hdnlc

BUCKET HEAD BACK LOCK COUNT
KERN_heap+65B9000 1 F10000F0059B93C0 F10000F0059B9240 00000000 16
KERN_heap+65B9018 2 F10000F0059B9600 F10000F0059B9660 00000000 1
KERN_heap+65B9288 28 F10000F0059C35C0 F10000F0059C3620 00000000 11
KERN_heap+65B9378 38 F10000F0059C6E00 F10000F0059C6F60 00000000 1
KERN_heap+65B9420 45 F10000F0059D9400 F10000F0059D9460 00000000 11
KERN_heap+65B9540 57 F10000F0059E1E00 F10000F0059E1E60 00000000 1
KERN_heap+65B9738 78 F10000F0059DB800 F10000F0059DB860 00000000 1
KERN_heap+65B9750 79 F10000F0059D6A00 F10000F0059D6A60 00000000 1
KERN_heap+65B9768 80 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B9810 87 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B9828 88 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B98A0 93 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B98D0 95 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B9900 97 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B9978 102 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B9990 103 F10000F0059D9400 F10000F0059D9460 00000000 1
KERN_heap+65B9A38 110 F10000F0059E1E00 F10000F0059E1E60 00000000 1
KERN_heap+65B9A80 113 F10000F0059E1E00 F10000F0059E1E60 00000000 1
KERN_heap+65B9B88 124 F10000F0059E1E00 F10000F0059E1E60 00000000 1
KERN_heap+65B9978

KDB Kernel debugger and kdb command

314
KDB(0)> hdnlc F1000F0065B9288
//effective address
HASH ENTRY( 28): F1000F0065B9288

DP     NP     NAME

443 KERN_heap+59C35C0 F1000F0049FB4B8 0000000000000000 __pv16.0
442 KERN_heap+59C3560 F1000F0049FB4B8 F1000F00557FFC8 __pv16.0
441 KERN_heap+59C3500 F1000F0049FB4B8 0000000000000000 __pv16.0
440 KERN_heap+59C34A0 F1000F0049FB4B8 F1000F0054E4FC8 __pv16.0
439 KERN_heap+59C3440 F1000F0049FB4B8 0000000000000000 __pv16.0
438 KERN_heap+59C33E0 F1000F0049FB4B8 F1000F00544A1F8 __pv16.0
437 KERN_heap+59C3380 F1000F0049FB4B8 0000000000000000 __pv16.0
436 KERN_heap+59C3320 F1000F0049FB4B8 F1000F0048C8B68 __pv16.0
435 KERN_heap+59C32C0 F1000F0049FB4B8 0000000000000000 __pv16.0
434 KERN_heap+59C3200 F1000F0049FB4B8 F1000F00557DA98 __pv16.0
433 KERN_heap+59C3200 F1000F0049FB4B8 0000000000000000 __pv16.0
432 KERN_heap+59C31A0 0000000000000000 0000000000000000
431 KERN_heap+59C3140 0000000000000000 0000000000000000
430 KERN_heap+59C30E0 0000000000000000 0000000000000000
429 KERN_heap+59C30C0 0000000000000000 0000000000000000
428 KERN_heap+59C3000 0000000000000000 0000000000000000
427 KERN_heap+59C2F80 0000000000000000 0000000000000000
426 KERN_heap+59C2F0 0000000000000000 0000000000000000
425 KERN_heap+59C2E8 0000000000000000 0000000000000000
424 KERN_heap+59C2E0 0000000000000000 0000000000000000
423 KERN_heap+59C2DC 0000000000000000 0000000000000000
422 KERN_heap+59C2C8 0000000000000000 0000000000000000
421 KERN_heap+59C2C0 0000000000000000 0000000000000000
420 KERN_heap+59C2B8 0000000000000000 0000000000000000
419 KERN_heap+59C2B0 0000000000000000 0000000000000000
418 KERN_heap+59C2A0 0000000000000000 0000000000000000
417 KERN_heap+59C298 0000000000000000 0000000000000000
416 KERN_heap+59C290 0000000000000000 0000000000000000
415 KERN_heap+59C288 0000000000000000 0000000000000000
414 KERN_heap+59C280 0000000000000000 0000000000000000
413 KERN_heap+59C278 0000000000000000 0000000000000000
412 KERN_heap+59C270 0000000000000000 0000000000000000
411 KERN_heap+59C268 0000000000000000 0000000000000000
410 KERN_heap+59C260 0000000000000000 0000000000000000
KDB(0)>
### kvn subcommand

#### Purpose
The `kvn` subcommand displays the `kdm` vnode data structure.

#### Syntax
`kvn address`

#### Parameters

`address` Identifies the address of the `kdm` vnode to display.

#### Aliases
No aliases.

#### Example
The following is an example of how to use the `kvn` subcommand:

```
KDB(0)> kvn 0x3173F180
kdv_enables..0x00000000 kdv_flags....0x00000000 kdv_nreg.....0x00000000
kdv_op........0x000000E0 kdv_fset......0x32F99400
kdv_regp.....0x00000000 kdv_data.....0x32F23628
```

**NOTE:** The `kdm` vnode pointer is in the JFS2 inode and may be obtained from the output of the `i2` command, in the `kdmvp` field:

```
KDB(0)> i2 32F23340
ADDRESS  DEVICE  I_NUM  IPMNT  COUNT  TYPE  FLAG
32F23340 002B0007 2 32F77020 00001  VDIR
```

In-memory Working Inode:
```
hashClass....0x000001B5 cacheClass....0x00000003 count........0x00000001
capability...0x0000069C3 atlhead.....0x00000000 atltail.....0x00000000
bxflag.......0x00000000 blid.........0x00000000 btindex.....0x00000000
diocnt.....0x000000001 nondioctn....0x00000000
dev...........0x002B0007 synctime.....0x00000000 nodelock.....0x00000000
ipmnt........0x32F77020 ipimap.......0x32F13340 pagerObject..0x00000000
event.......0xFFFFFFFF fsevent......0xFFFFFFFF openevent....0xFFFFFFFF
kdvList.ndxt.0x000000000 cachelst.prv.0x000000000 freeNext.....0x000000000
hashList.ndxt.0x32F93340 hashList.prv..0x31AA247C kdmvp........0x3173F180
flag.........0x00000000 cflag........0x00000000
xlock.........0x00000000 fsxlock.......0x00000000
btorder.....0x00000000
agstart.......0x0000000000000000 lastCommittedSize...0x0000000000000000
```
**buffer subcommand**

**Purpose**
The *buffer* subcommand displays buffer cache headers.

**Syntax**
```
buffer [slot | effectiveaddress]
```

**Parameters**
- **slot**: Specifies the buffer pool slot number. This parameter must be a decimal value.
- **effectiveaddress**: Specifies the effective address of a buffer pool entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

**Aliases**
- buf

**Example**
The following is an example of how to use the *buffer* subcommand:
```
KDB(0)> buf //print buffer pool
   1 057E4000 nodevice 00000000 00000000
   2 057E4058 nodevice 00000000 00000000
   3 057E40B0 nodevice 00000000 00000000
   4 057E4108 nodevice 00000000 00000000
   5 057E4160 nodevice 00000000 00000000
   ...
18 057E45D8 nodevice 00000000 00000000
19 057E4630 000A0011 00000000 00000100 READ DONE stale MPSAFE MPSAFE_INITIAL
20 057E4688 000A0011 00000000 00000008 READ DONE stale MPSAFE MPSAFE_INITIAL
KDB(0)> buf 19 //print buffer slot 19
   DEV VNODE BLKNO FLAGS

19 057E4630 000A0011 00000000 00000100 READ DONE stale MPSAFE MPSAFE_INITIAL
forw 0562F0CC back 0562F0CC av_forw 057E45D8 av_back 057E4688
blkno 00000100 addr 0580C000 bcount 00001000 resid 00000000
error 00000000 work 80000000 options 00000000 event FFFFFFFF
iodone: biodone+000000
start.tv_sec 00000000 start.tv_nsec 00000000
xmemd.aspace_id 00000000 xmemd.xm_flag 00000000 xmemd.xm_version 00000000
xmemd.subspace_id 00000000 xmemd.subspace_id2 00000000 xmemd.uaddr 00000000
KDB(0)> pdt 17 //print paging device slot 17 (the 1st FS)
```

PDT address B69C0440 entry 17 of 511, type: FILESYSTEM
next pdt on i/o list (nextio) : FFFFFFFF
dev_t or strategy ptr (device) : 000A0007
last frame w/pend I/O (iotal) : FFFFFFFF
free buf struct list (bufstr) : 056B2108
total buf structs (nbufs) : 005D
available (PAGING) (avail) : 0000
JFS disk agsize (agsize) : 0800
JFS inode agsize (iagsize) : 0800
JFS log SCB index (logidx) : 00035
JFS fragments per page(fperpage): 1
JFS compression type (comptype): 0
JFS log2 bigalloc mult(bigexp) : 0
disk map srval (dmsrval) : 00002021
i/o's not finished (iocnt) : 00000000
lock (lock) : EB003200

Chapter 30. Display general and Journal File System (JFS) information subcommands 317
buf 056B2108 //print paging device first free buffer

0 056B2108 000A0007 00000000 00000048 DONE SPLIT MPSAFE MPSAFE_INITIAL

forw 0007DAB3 back 00000000 av_forw 056B2080 av_back 00000000
blkno 00000048 addr 00000000 bcount 00001000 resid 00000000
error 00000000 work 00400000 options 00000000 event 00000000
iodone: v_pfend+000000
start.tv_sec 00000000 start.tv_nsec 00000000
xmemd.aspace_id 00000006 xmemd.xm_flag 00000000 xmemd.xm_version 00000000
xmemd.subspace_id 0083E01F xmemd.subspace_id2 00000000 xmemd.uaddr 00000000

KDB Kernel debugger and kdb command
**hbuffer subcommand**

**Purpose**
The *hbuffer* subcommand displays buffer cache hash list headers.

**Syntax**

```
hbuffer [bucket | effectiveaddress]
```

**Parameters**

- `bucket`  
  Specifies the bucket number of the buffer cache hash list entry. This parameter must be a decimal value.

- `effectiveaddress`  
  Specifies the effective address of a buffer cache hash list entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is specified, a summary for all entries is displayed. Display a specific entry by specifying the entry by bucket number or entry address.

**Aliases**

`hb`

**Example**

The following is an example of how to use the *hbuffer* subcommand:

```
KDB(0)> hb //print buffer cache hash lists
   BUCKET  HEAD  COUNT
 0562F0CC  18   057E4630   1
 0562F12C  26   057E4688   1
KDB(0)> hb 26 //print buffer cache hash list bucket 26
   DEV  VNODE  BLKNO  FLAGS
 20   057E4688 000A0011 00000000 00000008  READ DONE  STALE  MPSAFE  MPSAFE_INITIAL
```
fbuffer subcommand

Purpose
The fbuffer subcommand displays buffer cache freelist headers.

Syntax
fbuffer [bucket | effectiveaddress]

Parameters
bucket Specifies the bucket number of the buffer cache freelist entry. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of a buffer cache freelist entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is specified, a summary for all entries is displayed. Display a specific entry by specifying the entry by bucket number or entry address.

Aliases
fb

Example
The following is an example of how to use the fbuffer subcommand:

```
KDB(0)> fb  //print free list buffer buckets
  BUCKET  HEAD  COUNT
  bfreelist+000000 0001 057E4688  20
KDB(0)> fb 1  //print free list buffer bucket 1
  DEV  VNODE  BLKNO  FLAGS
  20 057E4688 000A0011 00000000 00000008 READ DONE STALE MPSAFE MPSAFE_INITIAL
  19 057E4630 000A0011 00000000 00000100 READ DONE STALE MPSAFE MPSAFE_INITIAL
  18 057E45D8 nodevice 00000000 00000000
  17 057E4580 nodevice 00000000 00000000
  ...
  2 057E4058 nodevice 00000000 00000000
  1 057E4000 nodedevice 00000000 00000000
```
gnode subcommand

Purpose
The gnode subcommand displays the generic node structure at the specified address.

Syntax

gnode effectiveaddress

Parameters

effectiveaddress Specifies the effective address of a generic node structure. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Aliases

gno

Example
The following is an example of how to use the gno alias for the gnode subcommand:

(8)> gno 09D0FD68 //print gnode
GNODE............ 09D0FD68
gn_type........ 00000002 gn_flags...... 00000000 gn_seg......... 0001A3FA
gn_mwrcnt...... 00000000 gn_mrdcnt...... 00000000 gn_rdcnt...... 00000000
gn_rwcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt...... 00000000
gn_vnode...... 0900FD28 gn_rdev...... 000A0010 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_recllk_lock. 00000000 gn_recllk_lock@ 09D0FD9C
gn_recllk_eventFFFFFFFF gn_filocks.... 00000000 gn_data....... 09D0FD58
gn_type....... DIR
gfs subcommand

Purpose
The gfs subcommand displays the generic file system structure at the specified address.

Syntax

gfs address

Parameters

address Specifies the address of a generic file system structure. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Aliases
No aliases.

Example
The following is an example of how to use the gfs subcommand:

(0)> gfs gfs //print gfs slot 1
gfs_data. 00000000 gfs_flag. INIT VERSION4 VERSION42 VERSION421
gfs_ops.. jfs_vfsopts gn_ops... jfs_vops gfs_name. jfs
gfs_init. jfs_init gfs_rinit jfs_rootinit gfs_type. JFS
gfs_hold. 00000012
(0)> gfs gfs+30 //print gfs slot 2
gfs_data. 00000000 gfs_flag. INIT VERSION4 VERSION42 VERSION421
gfs_ops.. spec_vfsops gn_ops... spec_vnops gfs_name. sfs
gfs_init. spec_init gfs_rinit nodev gfs_type. SFS
gfs_hold. 00000000
(0)> gfs gfs+60 //print gfs slot 3
gfs_data. 00000000 gfs_flag. REMOTE VERSION4
gfs_ops.. 01D2ABF8 gn_ops... 01D2A328 gfs_name. nfs
gfs_init. 01D2B5F0 gfs_rinit 00000000 gfs_type. NFS
gfs_hold. 0000000E
file subcommand

Purpose
The file subcommand displays file table entries.

Syntax
file [ slot | effectiveaddress]

Parameters
slot Specifies the slot number of a file table entry. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of a file table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the effective address.

If no parameter is used, all of the file table entries are displayed in a summary. Used files are displayed first. Detailed information can be displayed for individual file table entries by specifying the entry slot number or address.

Aliases
No aliases.

Example
The following is an example of how to use the file subcommand:

(0)> file //print file table
COUNT OFFSET DATA TYPE FLAGS
1 file+000000 1 0000000000000000 09CD90C8 VNODE EXEC
2 file+000030 1 0000000000000000 09CC2B50 VNODE EXEC
3 file+000060 1452 00000000000019B084 09CC2B50 VNODE READ RSHARE
4 file+000090 2 0000000000000100 09CFD80 VNODE EXEC
5 file+0000C0 2 0000000000000000 056CE008 VNODE READ WRITE
6 file+0000F0 1 0000000000000000 056CE008 VNODE READ WRITE
7 file+000120 1 0000000000000680 09CF680 VNODE READ WRITE
8 file+000150 1 0000000000000100 0978E0C VNODE EXEC
9 file+000180 2 0000000000000000 056CE070 VNODE READ NONBLOCK
10 file+0001B0 323 0000000000000061C 09CC4F30 VNODE READ RSHARE
11 file+0001E0 2 0000000000000000 0878E700 READ WRITE
12 file+000210 16 0000000000000000 03055AB8 VNODE READ RSHARE
13 file+000240 1 0000000000000000 0B21950 GNODE WRITE
14 file+000270 1 0000000000000000 0B21A20 GNODE WRITE
15 file+0002A0 2 000000000000005C 09CFFCE8 VNODE READ RSHARE
16 file+0002D0 2 0000000000000000 09CFF9B0 VNODE WRITE
17 file+000300 1 0000000000000000 0B78E800 READ WRITE
18 file+000330 1 0000000000000000 056CE008 VNODE READ
19 file+000360 1 0000000000000000 09CFB890 VNODE WRITE
20 file+000390 3 00000000000028A4 08B99A60C VNODE READ

(0)> more (^C to quit) ? Interrupted
(0)> file //print file slot 3
COUNT OFFSET DATA TYPE FLAGS
3 file+000060 1474 00000000000019B084 09CC2B50 VNODE READ RSHARE

f_flag........ 00001001 f_count......... 000005C2
f_mscount........ 0000 f_type............ 0001
f_data........ 09CC2B50 f_offset.... 000000000019B084
f_diroff...... 00000000 f_cred......... 056DE58
f_locof........ 004AF098 f_lock........ 00000000
f_offset_lock@. 004AF09C f_offset_lock@. 00000000
f_vinfo........ 00000000 f_ops........... 00250FC0 vnodeops+000000
VNODE......... 09CC2B50
v_flag.... 00000000 v_count... 00000002 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09CC2B5C v_vfsp.... 056D18A4
v_mvfsp... 00000000 v_gnode... 09CC2B90 v_next.... 00000000
v_vfsnext. 09CC2A08 vvfsprev. 09CC3968 vpfsnode 00000000
v_audit... 00000000
inode subcommand

Purpose
The inode subcommand displays inode table entries.

Syntax
inode [slot | effectiveaddress]

Parameters
slot Specifies the slot number of an inode table entry. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of an inode table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary for used inode table entries is displayed. The inode is considered used when count is greater than 0. Unused inodes are displayed with the fino subcommand. Detailed information is displayed for individual inode table entries by specifying the entry. The information is interpreted for special inodes. Special inodes include: mountnode and inodes.

Aliases
ino

Example
The following is an example of how to use the ino alias for the inode subcommand:

(0)> ino //print inode table

<table>
<thead>
<tr>
<th>DEV</th>
<th>NUMBER</th>
<th>CNT</th>
<th>GNODE</th>
<th>IPMNT</th>
<th>TYPE</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>02A4968</td>
<td>330003</td>
<td>10721</td>
<td>02A4978</td>
<td>09F79510</td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>02A9790</td>
<td>330003</td>
<td>10730</td>
<td>02A97A0</td>
<td>09F79510</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>02A32E90</td>
<td>330006</td>
<td>2948</td>
<td>02A32EA0</td>
<td>09F79510</td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>02A2ECD8</td>
<td>330006</td>
<td>3186</td>
<td>02A3CC90</td>
<td>09F79510</td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>09D01570</td>
<td>00A0005</td>
<td>14417</td>
<td>09D01580</td>
<td>09CC1990</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>09D7CE68</td>
<td>00A0005</td>
<td>47211</td>
<td>09D7CE78</td>
<td>09CC1990</td>
<td>REG, CHG, UPD, FSYNC, DIRTY</td>
<td></td>
</tr>
<tr>
<td>09D19C38</td>
<td>00A0005</td>
<td>6542</td>
<td>09D19C48</td>
<td>09CC1990</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>09D00238</td>
<td>00A0005</td>
<td>63718</td>
<td>09D00248</td>
<td>09CC1990</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>09D70918</td>
<td>00A0005</td>
<td>6746</td>
<td>09D70928</td>
<td>09CC1990</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>09D80000</td>
<td>00A0005</td>
<td>15184</td>
<td>09D80018</td>
<td>09CC1990</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>09F98450</td>
<td>0033003</td>
<td>4098</td>
<td>09F98460</td>
<td>09F79510</td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>09F996D8</td>
<td>0033003</td>
<td>4097</td>
<td>09F996E8</td>
<td>09F79510</td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>09C6548</td>
<td>0033006</td>
<td>4110</td>
<td>09C6558</td>
<td>09F79510</td>
<td>DIR</td>
<td></td>
</tr>
<tr>
<td>09F83008</td>
<td>00330005</td>
<td>4104</td>
<td>09F830E8</td>
<td>09F7950</td>
<td>DIR, CHG, UPD, FSYNC, DIRTY</td>
<td></td>
</tr>
<tr>
<td>09FAB888</td>
<td>0033003</td>
<td>4117</td>
<td>09FAB878</td>
<td>09F79510</td>
<td>REG</td>
<td></td>
</tr>
<tr>
<td>0A492A88</td>
<td>0033003</td>
<td>4123</td>
<td>0A492AC8</td>
<td>09F79510</td>
<td>REG</td>
<td></td>
</tr>
</tbody>
</table>

(0)> more (^C to quit)? //Interrupted

(0)> ino 09F79510 //print mount table inode (IPMNT)

<table>
<thead>
<tr>
<th>DEV</th>
<th>NUMBER</th>
<th>CNT</th>
<th>GNODE</th>
<th>IPMNT</th>
<th>TYPE</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>09F79510</td>
<td>00330003</td>
<td>0</td>
<td>09F79520</td>
<td>09F79510</td>
<td>NON, CMNEW</td>
<td></td>
</tr>
</tbody>
</table>

forw 09F78C18 back 09F7A588 next 09F79510 prev 09F79510
gnode 09F79520 number 00000000 dev 00330003 ipmnt 09F79510
flag 00000000 locks 00000000 bigexp 00000000 compress 00000000
cflag 00000002 count 00000001 event FFFFFFFF movedfrag 00000000

Chapter 30. Display general and Journal File System (JFS) information subcommands 325
openevent FFFFFF id 000052A8 hip 09C9C330 nodelock 00000000
nodelock@ 09F79590 dquot[USR]00000000 dquot[GRP]00000000 dinode@ 09F7959C
cluster 00000000 size 0000000000000000

GNODE........... 09F79520
gn_type........ 00000000 gn_flags...... 00000000 gn_seg.......... 00000000
gn_mwrcnt...... 00000000 gn_mrdcnt...... 00000000 gn_rdcnt...... 00000000
gn_wrcrent...... 00000000 gn_excnt...... 00000000 gn_rshcnt...... 00000000
gn_viode...... 09F799400 gn_rdev...... 00000000 gn_ops........ jfs_vops
gn_chan...... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09F79554
gn_reclk_event FFFFFF gn_filocks.... 00000000 gn_data...... 09F79510
gn_type....... NON
di_gen 32B69977 di_mode 00000000 di_nlink 00000000
di_acct 00000000 di_uid 00000000 di_gid 00000000
di_nbblocks 00000000 di_acl 00000000
di_mtime 00000000 di_atime 00000000 di_ctime 00000000
di_size_hi 00000000 di_size_lo 00000000

VNODE........... 09F794E0
v_flag.... 00000000 v_count... 00000000 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09F79520 v_lock@... 09F79520
v_vfsnext. 00000000 v_vfsnext. 00000000 v_vfsprev. 00000000
v_audit... 00000000
di_iplog 09F77F48 di_ipinode 09F798E8 di_ipind 09F797A0
di_ipinomap 09F799A30 di_ipmap 09F79B78 di_ipsuper 09F79658
di_ipinodex 09F79CC0 di_ipsuper 09F79658
di_jmpmnt 00000000 di_jmpmnt 00000000
(di) > ino 09F77F48 //print log inode (di_iplog)
  DEV NUMBER CNT GNODE IPMNT TYPE FLAGS
          09F77F48 00330001  0 5 09F77F58 09F77F48 NON CMNEW

forw 09C9C310 back 09F785B0 next 09F77F48 prev 09F77F48
gnode@ 09F77F58 number 00000000 dev 00330001 ipmnt 09F77F48
flag 00000000 locks 00000000 bigexp 00000000 compress 00000000
cflag 0000002 count 00000000 event FFFFFF movedfrag 00000000
openevent FFFFFF id 0000029A hip 09C9C310 nodelock 00000000
nodelock@ 09F797C8 dquot[USR]00000000 dquot[GRP]00000000 dinode@ 09F77FD4
cluster 00000000 size 0000000000000000

GNODE........... 09F77F58
gn_type........ 00000000 gn_flags...... 00000000 gn_seg.......... 000007547
gn_mwrcnt...... 00000000 gn_mrdcnt...... 00000000 gn_rdcnt...... 00000000
gn_wrcrent...... 00000000 gn_excnt...... 00000000 gn_rshcnt...... 00000000
gn_viode...... 09F77F18 gn_rdev...... 00000000 gn_ops........ jfs_vops
gn_chan...... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09F77F0C
gn_reclk_event FFFFFF gn_filocks.... 00000000 gn_data...... 09F77F48
gn_type....... NON
di_gen 32B69976 di_mode 00000000 di_nlink 00000000
di_acct 00000000 di_uid 00000000 di_gid 00000000
di_nbblocks 00000000 di_acl 00000000
di_mtime 00000000 di_atime 00000000 di_ctime 00000000
di_size_hi 00000000 di_size_lo 00000000

VNODE........... 09F77F18
v_flag.... 00000000 v_count... 00000000 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09F77F24 v_lock@... 09F77F24
v_vfsnext. 00000000 v_vfsnext. 00000000 v_vfsprev. 00000000
v_audit... 00000000

326 KDB Kernel debugger and kdb command
Chapter 30. Display general and Journal File System (JFS) information subcommands

```
di_logptr 0000015A di_logsize 00000C00 di_logend 00000FF8
di_logsync 0005A994 di_nextsync 0013BBFC di_logxor 6CB68513
di_llogeor 00000FE0 di_llogxor 6CE29103 di_logx 0BB13200
di_logdgp 0B7E5BC0 di_loglock 4004B9EF di_loglock0 09F7804C
logxlock 00000000 logxlock0 0BB13200 logflag 00000001
logpong 00000195 logcq.head B69CA87C logcq.tail 0BB1322B
logcsn 00001534 logcrtc 0000000C loglcrct B69CA97C
logeopm 00000000 logeopmc 00000002
logeopmq[0]@ 0BB1322B logeopmq[1]@ 0BB1322B
```
**hinode subcommand**

**Purpose**
The *hinode* subcommand displays inode hash list entries.

**Syntax**

`hinode [bucket | address]`

**Parameters**

- **bucket**
  Specifies the bucket number. This parameter must be a decimal value.

- **address**
  Specifies the effective address of an inode hash list entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, the hash list is displayed. View the entries for a specific hash table entry by specifying a bucket number or the address of a hash list bucket.

**Aliases**

- **hino**

**Example**

The following is an example of how to use the *hino* alias for the *hinode* subcommand:

(0)> hino //print hash inode buckets

<table>
<thead>
<tr>
<th>BUCKET</th>
<th>HEAD</th>
<th>TIMESTAMP</th>
<th>LOCK</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>09C86000</td>
<td>1</td>
<td>0A285470</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86010</td>
<td>2</td>
<td>0A28AE08</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86020</td>
<td>3</td>
<td>0A2843C8</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86030</td>
<td>4</td>
<td>0A287E88</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86040</td>
<td>5</td>
<td>0A287330</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86050</td>
<td>6</td>
<td>0A2867A8</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86060</td>
<td>7</td>
<td>0A285FF8</td>
<td>00000007</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86070</td>
<td>8</td>
<td>0A289D78</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86080</td>
<td>9</td>
<td>0A289858</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86090</td>
<td>10</td>
<td>0A33E2D8</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C860A0</td>
<td>11</td>
<td>0A33E7F8</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C860B0</td>
<td>12</td>
<td>0A33EE60</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C860C0</td>
<td>13</td>
<td>0A33F758</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C860D0</td>
<td>14</td>
<td>0A28AE20</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C860E0</td>
<td>15</td>
<td>0A28A670</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C860F0</td>
<td>16</td>
<td>0A33CE58</td>
<td>00000005</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86100</td>
<td>17</td>
<td>0A339E00</td>
<td>00000006</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86110</td>
<td>18</td>
<td>0A5FF6D0</td>
<td>00000008</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86120</td>
<td>19</td>
<td>0A5FD060</td>
<td>00000009</td>
<td>00000000</td>
</tr>
<tr>
<td>09C86130</td>
<td>20</td>
<td>0A5FC390</td>
<td>00000009</td>
<td>00000000</td>
</tr>
</tbody>
</table>

(0)> more (^C to quit) ? Interrupted

(0)> hino 18 //print hash inode bucket 18

**HASH ENTRY( 18): 09C86110**

<table>
<thead>
<tr>
<th>DEV</th>
<th>NUMBER</th>
<th>CNT</th>
<th>GNODE</th>
<th>IPMNT</th>
<th>TYPE</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A5FF6D0</td>
<td>00330003</td>
<td>2523</td>
<td>0</td>
<td>0A5FF6E0</td>
<td>09F79510</td>
<td>REG</td>
</tr>
<tr>
<td>0A340E68</td>
<td>00330004</td>
<td>2524</td>
<td>0</td>
<td>0A340E78</td>
<td>09F78090</td>
<td>REG</td>
</tr>
<tr>
<td>0A28B0A60</td>
<td>00330006</td>
<td>10677</td>
<td>0</td>
<td>0A28B0A60</td>
<td>09F79510</td>
<td>DIR</td>
</tr>
<tr>
<td>0A1AF0A0</td>
<td>00330006</td>
<td>2526</td>
<td>0</td>
<td>0A1AF0A0</td>
<td>09F7A990</td>
<td>REG</td>
</tr>
</tbody>
</table>
icache subcommand

Purpose
The icache subcommand displays inode cache list entries.

Syntax
icache [slot | effectiveaddress]

Parameters
slot Specifies the slot number of an inode cache list entry. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of an inode cache list entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed. Display detailed information for a particular entry by specifying the entry to display with either the slot number or the address.

Aliases
fino

Example
The following is an example of how to use the fino alias for the icache subcommand:
(0)> fino //print free inode cache

DEV NUMBER CNT GNODE IPMNT TYPE FLAGS
  1 09CABFA0 DEADBEEF 0 0 09CABFB0 09CA717B CHR CMNOLINK
  2 0A8D3A70 DEADBEEF 0 0 0A8D3A80 09F7A990 REG CMNOLINK
  3 0A8F2528 DEADBEEF 0 0 0A8F2538 09CC652B REG CMNOLINK
  4 0A7C66E0 DEADBEEF 0 0 0A7C66F0 09F7A990 REG CMNOLINK
  5 0A78A568 DEADBEEF 0 0 0A78A578 09F7F50 REG CMNOLINK
  6 0A78EC68 DEADBEEF 0 0 0A78EC78 09F8009 REG CMNOLINK
  7 0A7AF9B8 DEADBEEF 0 0 0A7AF9C8 09F7F50 REG CMNOLINK
  8 0A789230 DEADBEEF 0 0 0A789240 09F7F50 REG CMNOLINK
  9 0A8BDC8C DEADBEEF 0 0 0A8BDCBC 09F7A990 REG CMNOLINK
 10 0A8BEB98 DEADBEEF 0 0 0A8BEBB8 09F7A990 REG CMNOLINK
 11 0A7C58C8 DEADBEEF 0 0 0A7C58D8 09F7A990 REG CMNOLINK
 12 0A78D6A0 DEADBEEF 0 0 0A78D6B0 09F8009 REG CMNOLINK
 13 0A7C4BF8 DEADBEEF 0 0 0A7C4CB8 09F7A990 REG CMNOLINK
 14 0A78AD1A DEADBEEF 0 0 0A78AD20 09F8009 REG CMNOLINK
 15 0A78B8A0 DEADBEEF 0 0 0A78B8A0 09F7F50 REG CMNOLINK
 16 0A8BC970 DEADBEEF 0 0 0A8BC980 09F7A990 REG CMNOLINK
 17 0A8010F8 DEADBEEF 0 0 0A8010F8 09F7A990 REG CMNOLINK
 18 0A7AE200 DEADBEEF 0 0 0A7AE208 09F7F50 REG CMNOLINK
 19 0A8EF9F8 DEADBEEF 0 0 0A8EF9F8 09CC652B REG CMNOLINK
 20 0A7C418B DEADBEEF 0 0 0A7C419B 09F7A990 REG CMNOLINK

(0)> more (^C to quit) ? Interrupted
(0)> fino //print free inode slot 1

DEV NUMBER CNT GNODE IPMNT TYPE FLAGS
  09CABFA0 DEADBEEF 0 0 09CABFB0 09CA717B CHR CMNOLINK

forw 09CABFA0 back 09CABFA0 next 0A8EF70B prev 0042AE60
gnode@ 09CABFB0 number 00000000 dev DEADBEEF ipmnt 09CA717B
flag 00000000 locks 00000000 bigexp 00000000 compress 00000000
cflag 00000004 count 00000000 event FFFFFFFFFFFF movedfrag 00000000
openevent FFFFFFFFFFFF id 00000045 hip 00000000 nodelock 00000000
nodelock@ 09CAC020 dquot[USR]00000000 dquot[GRP]00000000 dinode@ 09CAC020C

Chapter 30. Display general and Journal File System (JFS) information subcommands 329
cluster 00000000 size 0000000000000000

GNODE........... 09CABFB0
  gn_type........ 00000004 gn_flags...... 00000000 gn_seg......... 00000000
  gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt....... 00000000
  gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt...... 00000000
  gn_vnode...... 09CABF70 gn_rdev...... 00030000 gn_ops........ jfs_vops
  gn_chan........ 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 09CABFE4
  gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 09CABFA0
  gn_type....... CHR
di_gen 00000000 di_mode 00000000 di_nlink 00000000
di_acct 00000000 di_uid 00000000 di_gid 00000000
di_nbblocks 00000000 di_acl 00000000
di_mtime 32B67A97 di_atime 32B67A97 di_ctime 32B67B4B
di_size_hi 00000000 di_size_lo 00000000
di_rdev 00030000

VNODE........... 09CABF70
  v_flag.... 00000000 v_count... 00000000 v_vfsgen.. 00000000
  v_lock.... 00000000 v_lock@... 09CABF7C v_vfsp.... 00000000
  v_mvfsps.. 00000000 v_gnode... 09CABFB0 v_next.... 00000000
  v_vfsnext. 09CABE26 v_vfsprev. 00000000 v_pfsvnode 00000000
  v_audit... 00000000

330 KDB Kernel debugger and kdb command
vnc subcommand

Purpose
The vnc subcommand displays information about the vnode cache filesystem.

Syntax
vnc [slot | effectiveaddress]

Parameters

- **slot**: Specifies the decimal identifier of a specific cache entry.
- **effectiveaddress**: Specifies the effective address of the entry.

You can only specify one parameter.

When no parameters are provided, a summary of the entire vnode cache is displayed. If there are no valid cache entries in memory, nothing is displayed.

Aliases
vcache

Example
The following is an example of how to use the vnc subcommand:

```bash
(0)> vnc

<table>
<thead>
<tr>
<th>VP</th>
<th>DEV</th>
<th>INO</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F10010F0109A</td>
<td>F10010F007F929B0</td>
<td>8000000A00000000B</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F10010F0109A</td>
<td>F10010F0109A0060</td>
<td>00000000000000000</td>
<td>80000021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F10010F0109A</td>
<td>F10010F0109A0030</td>
<td>F10010F0109A0090</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>F10010F0109A</td>
<td>F10010F0109A0150</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>F10010F0109A</td>
<td>F10010F0109A0120</td>
<td>F10010F0109A0180</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>F10010F0109A</td>
<td>F10010F0109A0240</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>F10010F0109A</td>
<td>F10010F0109A0270</td>
<td>F10010F0109A0210</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>F10010F0109A</td>
<td>F10010F0109A0330</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>F10010F0109A</td>
<td>F10010F0109A0360</td>
<td>F10010F0109A0320</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>F10010F0109A</td>
<td>F10010F0109A0420</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>F10010F0109A</td>
<td>F10010F0109A0450</td>
<td>F10010F0109A0410</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>F10010F0109A</td>
<td>F10010F0109A0510</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>F10010F0109A</td>
<td>F10010F0109A0540</td>
<td>F10010F0109A0500</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>F10010F0109A</td>
<td>F10010F0109A00000</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>F10010F0109A</td>
<td>F10010F0109A0800</td>
<td>F10010F0121A0018</td>
<td>F10010F0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>F10010F0109A</td>
<td>F10010F0109A0618</td>
<td>6BE7617000000000</td>
<td>F10010F0121A0018</td>
<td>F10010F0</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>F10010F0109A</td>
<td>F10010F0109A06F0</td>
<td>00000000000000000</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>F10010F0109A</td>
<td>F10010F0109A06C0</td>
<td>F10010F0109A06E0</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>F10010F0109A</td>
<td>F10010F0109A0720</td>
<td>F10010F0109A07A0</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>F10010F0109A</td>
<td>F10010F0109A0780</td>
<td>F10010F0109A07D0</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(0)> more (^C to quit) ? //Interrupted

(0)> vnc 1

<table>
<thead>
<tr>
<th>VP</th>
<th>DEV</th>
<th>INO</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F10010F0109A</td>
<td>F10010F007F929B0</td>
<td>8000000A00000000B</td>
<td>00000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>forw.....</td>
<td>F10010F0109A05D0</td>
<td>back.....</td>
<td>F10010F0121A0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ino.......</td>
<td>00000000</td>
<td>gen......</td>
<td>0000C125</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v_flag....</td>
<td>00000000</td>
<td>v_count...</td>
<td>00000001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v_vfsgend</td>
<td>00000000</td>
<td>v_vfsp.....</td>
<td>F10010F00EE42940</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v_lock@...</td>
<td>F10010F007F929C0</td>
<td>v_lock....</td>
<td>0000000000000000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 30. Display general and Journal File System (JFS) information subcommands 331
v_mvfsp... 0000000000000000 v_gnode... F10010F007F92A28
v_next.... 0000000000000000 v_vfsnext. F10010F007D629B0
v_vfsprev. F10010F0081C29B0 v_pfsvnode 0000000000000000
v_audit... 0000000000000000
**hvnc subcommand**

**Purpose**
The hvnc subcommand displays information about the filesystem hash list for the vnode cache.

**Syntax**
```
hvnc [slot | effectiveaddress]
```

**Parameters**
- `slot` Specifies the decimal identifier of a specific hash bucket.
- `effectiveaddress` Specifies the effective address of the entry. Symbols, hexadecimal values, or hexadecimal expressions can be used in specification of the address.

The hvnc command is used to display information about the vcache hash table. When no parameters are provided, a summary of the entire hash list is displayed.

**Aliases**
hvcache

**Example**
The following is an example of how to use the hvnc subcommand:

```
(0)> hvnc

<table>
<thead>
<tr>
<th>BUCKET</th>
<th>HEAD</th>
<th>BACK</th>
<th>LOCK</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>F10010F01210000</td>
<td>1</td>
<td>F10010F0109A0000</td>
<td>F10010F0109A0030</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210016</td>
<td>2</td>
<td>F10010F0109A0060</td>
<td>F10010F0109A0050</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210060</td>
<td>5</td>
<td>F10010F0109A1800</td>
<td>F10010F0109A1830</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210078</td>
<td>6</td>
<td>F10010F0109A1E00</td>
<td>F10010F0109A1E30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F012100C0</td>
<td>9</td>
<td>F10010F0109A3000</td>
<td>F10010F0109A3030</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F012100F0</td>
<td>11</td>
<td>F10010F0109A4200</td>
<td>F10010F0109A4230</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210138</td>
<td>14</td>
<td>F10010F0109A4E00</td>
<td>F10010F0109A4E30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210150</td>
<td>15</td>
<td>F10010F0109A5400</td>
<td>F10010F0109A5430</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210168</td>
<td>16</td>
<td>F10010F0109A5A00</td>
<td>F10010F0109A5A30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210180</td>
<td>19</td>
<td>F10010F0109A6C00</td>
<td>F10010F0109A6C30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F012101C8</td>
<td>20</td>
<td>F10010F0109A7230</td>
<td>F10010F0109A7260</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F012101DE</td>
<td>21</td>
<td>F10010F0109A7800</td>
<td>F10010F0109A7830</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F012101F8</td>
<td>22</td>
<td>F10010F0109A7E00</td>
<td>F10010F0109A7E30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210228</td>
<td>24</td>
<td>F10010F0109A8A00</td>
<td>F10010F0109A8A30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210240</td>
<td>25</td>
<td>F10010F0109A9060</td>
<td>F10010F0109A9090</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210258</td>
<td>26</td>
<td>F10010F0109A9600</td>
<td>F10010F0109A9630</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210270</td>
<td>27</td>
<td>F10010F0109A9C00</td>
<td>F10010F0109A9C30</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F01210288</td>
<td>30</td>
<td>F10010F0109AAE30</td>
<td>F10010F0109AAE60</td>
<td>00000000</td>
</tr>
<tr>
<td>F10010F012102DD</td>
<td>31</td>
<td>F10010F0109AB400</td>
<td>F10010F0109AB430</td>
<td>00000000</td>
</tr>
</tbody>
</table>

(0)> more (^C to quit) //Interrupted

(0)> hvnc 1

HASH ENTRY(  1): F10010F01210000

<table>
<thead>
<tr>
<th></th>
<th>VP</th>
<th>DEV</th>
<th>INO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F10010F0109A0000</td>
<td>F10010F0107F92980</td>
<td>00000000A0000000B</td>
</tr>
<tr>
<td>38</td>
<td>F10010F0109A05D0</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>37</td>
<td>F10010F0109A05A0</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>35</td>
<td>F10010F0109A0570</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>34</td>
<td>F10010F0109A0540</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>33</td>
<td>F10010F0109A0510</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>32</td>
<td>F10010F0109A04E0</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>

Chapter 30. Display general and Journal File System (JFS) information subcommands 333
31 F10010F0109A04B0 0000000000000000 0000000000000000 00000000
29 F10010F0109A0480 0000000000000000 0000000000000000 00000000
28 F10010F0109A0450 0000000000000000 0000000000000000 00000000
27 F10010F0109A0420 0000000000000000 0000000000000000 00000000
26 F10010F0109A03F0 0000000000000000 0000000000000000 00000000
25 F10010F0109A03C0 0000000000000000 0000000000000000 00000000
23 F10010F0109A0390 0000000000000000 0000000000000000 00000000
22 F10010F0109A0360 0000000000000000 0000000000000000 00000000
21 F10010F0109A0330 0000000000000000 0000000000000000 00000000
20 F10010F0109A0300 0000000000000000 0000000000000000 00000000
19 F10010F0109A02D0 0000000000000000 0000000000000000 00000000
17 F10010F0109A02A0 0000000000000000 0000000000000000 00000000
16 F10010F0109A0270 0000000000000000 0000000000000000 00000000

(0)>
**vnode subcommand**

**Purpose**
The *vnode* subcommand displays virtual node (vnode) table entries.

**Syntax**
```
vnode [slot | effectiveaddress]
```

**Parameters**
- **slot** Specifies the slot number of a virtual node table entry. This parameter must be a decimal value.
- **effectiveaddress** Specifies the effective address of a virtual node table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed with one line per table entry. Display detailed information for individual vnode table entries by specifying the entry with either a slot number or an address.

**Aliases**
- vno

**Example**
The following is an example of how to use the *vnode* subcommand:

```
(0)> vnode //print vnode table
COUNT VFSGEN GNODE VFSP DATAPTR TYPE FLAGS
106 09D227B0 3 0 09D227F0 056D183C 00000000 REG
126 09D1A868 1 0 09D1A8A8 056D183C 00000000 REG
130 09D196E8 1 0 09D19728 056D183C 00000000 REG
135 09D188E0 1 0 09D188A0 056D183C 05CC2D00 SOCK
140 09D17E70 1 0 09D17ED0 056D183C 05CC2D00 SOCK
143 09D17970 1 0 09D179B0 056D183C 05CC2A00 SOCK
148 09D17078 1 0 09D170B8 056D183C 05D3F300 SOCK
154 09D14DE0 1 0 09D14E20 056D183C 00000000 REG
162 09D0D948 1 0 09D0D988 056D183C 00000000 DIR
165 09D0D050 1 0 09D0D090 056D183C 05030E00 SOCK
167 09D0D688 1 0 09D0D6A0 056D183C 00000000 DIR
170 09D0D2E0 1 0 09D0D320 056D183C 00000000 DIR
171 09D0D198 1 0 09D0D1D0 056D183C 00000000 DIR
172 09D0D570 1 0 09D0D5B0 056D183C 00000000 DIR
173 09D0C0F8 1 0 09D0CF40 056D183C 00000000 DIR
174 09D0CC00 1 0 09D0CE00 056D183C 00000000 DIR
175 09D0CC78 1 0 09D0CCE8 056D183C 00000000 DIR
176 09D0CB30 1 0 09D0CB70 056D183C 00000000 DIR
```

```
(0)> more (^C to quit) //Interrupted
(0)> vnode 106 //print vnode slot 106
COUNT VFSGEN GNODE VFSP DATAPTR TYPE FLAGS
106 09D227B0 3 0 09D227F0 056D183C 00000000 REG
```

```
v_flag.... 00000000 v_count... 00000003 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 09D2278C v_vfsp.... 056D183C
v_mvfsp... 00000000 v_gnode... 09D227f0 v_next.... 00000000
v_vfsnext. 09D22668 v_vfsprev. 09D22688 v_psvnode 00000000
v_audit... 00000000
```
**vfs subcommand**

**Purpose**
The `vfs` subcommand displays entries of the virtual file system table.

**Syntax**
`vfs [slot | address]`

**Parameters**
- `slot` Specifies the slot number of a virtual file system table entry. This parameter must be a decimal value.
- `address` Specifies the address of a virtual file system table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed with one line for each entry. Display detailed information by identifying the entry of interest with either a slot number or an address.

**Aliases**
`mount`

**Example**
The following is an example of how to use the `vfs` subcommand:

```
(0)> vfs //print vfs table

<table>
<thead>
<tr>
<th>GFS</th>
<th>MNTD</th>
<th>MNTDOVER</th>
<th>VNODES</th>
<th>DATA</th>
<th>TYPE</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>056D183C</td>
<td>0024F268</td>
<td>09CC08B8</td>
<td>00000000</td>
<td>0A5AA0A0</td>
<td>0B221F68</td>
</tr>
<tr>
<td>...</td>
<td>/dev/hd4 mounted over /</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>056D18A4</td>
<td>0024F268</td>
<td>09CC2258</td>
<td>09CC08B8</td>
<td>0A545270</td>
<td>0B221F00</td>
</tr>
<tr>
<td>...</td>
<td>/dev/hd2 mounted over /var</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>056D1870</td>
<td>0024F268</td>
<td>09CC3820</td>
<td>09CC2258</td>
<td>09D913A8</td>
<td>0B221E30</td>
</tr>
<tr>
<td>...</td>
<td>/dev/hd9var mounted over /var</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>056D1808</td>
<td>0024F268</td>
<td>09CC60F0</td>
<td>09CC3820</td>
<td>0A72C1E8</td>
<td>0B221818</td>
</tr>
<tr>
<td>...</td>
<td>/dev/hd3 mounted over /tmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>056D1808</td>
<td>0024F268</td>
<td>09D0BF8</td>
<td>09008568</td>
<td>09D95500</td>
<td>0B2412F0</td>
</tr>
<tr>
<td>...</td>
<td>/dev/hd1 mounted over /home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>056D189C</td>
<td>0024F2C8</td>
<td>082430C0</td>
<td>0900C238</td>
<td>080F6A0C</td>
<td>0B230500</td>
</tr>
<tr>
<td>...</td>
<td>/pvt/tools mounted over /pvt/tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>056D1940</td>
<td>0024F2C8</td>
<td>087E4A0C</td>
<td>0900C830</td>
<td>08934C0C</td>
<td>0B230A00</td>
</tr>
<tr>
<td>...</td>
<td>/pvt/base mounted over /pvt/base</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>056D1974</td>
<td>0024F2C8</td>
<td>087E4A0C</td>
<td>0900C78</td>
<td>087E4A0C</td>
<td>0B230C00</td>
</tr>
<tr>
<td>...</td>
<td>/pvt/perm mounted over /pvt/perm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>056D19A8</td>
<td>0024F2C8</td>
<td>087E4D0C</td>
<td>0900DCC0</td>
<td>0B99000C</td>
<td>0B230E00</td>
</tr>
<tr>
<td>...</td>
<td>/nfs mounted over /nfs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>056D19DC</td>
<td>0024F2C8</td>
<td>0899200C</td>
<td>0900CF08</td>
<td>0B89840C</td>
<td>0B230000</td>
</tr>
<tr>
<td>...</td>
<td>/tcp mounted over /tcp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(0)> vfs 5 //print vfs slot 5

<table>
<thead>
<tr>
<th>GFS</th>
<th>MNTD</th>
<th>MNTDOVER</th>
<th>VNODES</th>
<th>DATA</th>
<th>TYPE</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>056D1808</td>
<td>0024F268</td>
<td>09D0BF8</td>
<td>09008568</td>
<td>09D95500</td>
<td>0B2412F0</td>
</tr>
<tr>
<td>...</td>
<td>/dev/hd1 mounted over /home</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
vfs.next.... 056D190C vfs_count.... 00000001 vfs_mntd.... 09D0BF8
vfs.mntdover. 09008568 vfs_vnodes.... 09D95500 vfs_count.... 00000001
cfs_number... 00000009 vfs_bsize.... 00010000 vfs_mdatal... 0B7E8EB0
vm.revision. 00000001 vm_length... 00000070 vfs_fsid.... 00000008 00000003
cfs_vfsws... 00000009 vfs_date.... 32B67BFF vfs_flag.... 00000004
vm_gfstype... 00000003 @vm_data.... 0B7E8E4A vfs_lock.... 00000000
```
Chapter 30. Display general and Journal File System (JFS) information subcommands

VFS_GFS... gfs+000000
gfs_data... 00000000 gfs_flag... INIT VERSION4 VERSION42 VERSION421
gfs_ops... jfs_vfsops gn_ops... jfs_vops gfs_name... jfs
gfs_init... jfs_init gfs_rinit... jfs_rootinit gfs_type... JFS
gfs_hold... 00000013

VFS_MNTO... 09D0BFA8
v_flag... 00000001 v_count... 00000001 v_vfsgen... 00000000
v_lock... 00000000 v_lock@... 09D0BF38 v_fsp... 056D18D8
v_mvfsp... 00000000 v_gnode... 09D0BF38 v_next... 00000000
v_vfsnext... 00000000 v_vfsprev... 09D730A0 v_pfsnode... 00000000
v_audit... 00000000 v_flag... ROOT

VFS_MNTO... 09D0BFA8
v_flag... 00000001 v_count... 00000001 v_vfsgen... 00000000
v_lock... 00000000 v_lock@... 09D0BF38 v_fsp... 056D18D8
v_mvfsp... 056D18D8 v_gnode... 09D0BF38 v_next... 00000000
v_vfsnext... 09D0A230 v_vfsprev... 09D0C0F0 v_pfsnode... 00000000
v_audit... 00000000

VFS_VNODES LIST...
COUNT VFSGEN GNODE VFSP DATAPTR TYPE FLAGS
1 09D95500 0 0 09D955A0 056D18D8 00000000 REG
2 09D94AC0 0 0 09D94B00 056D18D8 00000000 DIR
3 09D91DE8 0 0 09D91E28 056D18D8 00000000 REG
4 09D91A10 0 0 09D91A50 056D18D8 00000000 DIR
5 09D9EFC8 0 0 09D8F008 056D18D8 00000000 REG
6 09D8BEB0 0 0 09D8EC30 056D18D8 00000000 DIR
7 09D8C5B0 0 0 09D8C5CB 056D18D8 00000000 REG
8 09D8CO60 0 0 09D8COA0 056D18D8 00000000 DIR
9 09D8A058 0 0 09D8A098 056D18D8 00000000 REG
10 09D89BC0 0 0 09D89CC0 056D18D8 00000000 DIR
11 09D892A0 0 0 09D892B0 056D18D8 00000000 REG
...
COUNT VFSGEN GNODE VFSP DATAPTR TYPE FLAGS
63 09D73478 0 0 09D734B8 056D18D8 00000000 REG
64 09D730A0 0 0 09D730E0 056D18D8 00000000 DIR
65 09D8BFA8 1 0 09D8FEB8 056D18D8 00000000 DIR ROOT
**specnode subcommand**

**Purpose**
The `specnode` subcommand displays the special device node structure at the specified address.

**Syntax**

```
specnode address
```

**Parameters**

`address` Specifies the effective address of a special device node structure. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

**Aliases**

`specno`

**Example**
The following is an example of how to use the `specno` alias for the `specnode` subcommand:

```
KDB(0)> file 108 //print file entry
        ADDR  COUNT  OFFSET  DATA  TYPE  FLAGS
108   10001410  1  0000000000000000 32ABD1DC VNODE WRITE NOCTTY

f_flag........ 00000002 f_count........ 00000001
f_options...... 0000 f_type........... 0001
f_data........ 32ABD1DC f_offset... 0000000000000000
f_dir off...... 00000000 f_cred........ 32BB5600
f_lock@........ 10001430 f_lock...... 00000000
f_offset lock@. 10001434 f_offset ...... 00000000
f_vinfo....... 00000000 f_ops........ 006A2F98 vnodefops+000000
VNODE......... 32ABD1DC
v_flag........ 00000000 v_count... 000000018 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 32AB01E8 v_vfsp.... 01FB4000
v_mvfsp... 00000000 v_gnode... 32843080 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfs vnode 14546080
v_auditt... 00000000
KDB(0)> gno 32843080 //print gnode node entry
GNODE.......... 32843080 32843080

gn_type...... 00000009 gn_flags...... 00000000 gn_seg........ 007FFFFFFF
gn_mwrcnt.... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt..... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 32ABD1DC gn_rdev...... 00400000 gn_ops........ spec_vnops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 32843084
gn_reclk_eventFFFFFFFFFF gn_filocks.... 00000000 gn_data.... 32843070
gn_type...... MPC
KDB(0)> specno 32843070 //print special node entry
SPECIALNODE.... 32843070

sn_next...... 00000000 sn_gen....... 00000537 sn_count.... 0001
sn_gnode.... 032843080 sn_pfsnode.. 145460C0 sn_lock..... 03284307C 00000000
sn_atrr..... 328560C0 sn_dev........ 00000000 sn_chan..... 00000000
sn_vnode..... 32AB01DC sn_ops....... 06D99990 sn_type...... 00000009
sn_data...... 328439A8 fdev_chain_f. 00000000 sn_type..... MPC
sn_mode...... 0002192 sn_uid....... 00000000 sn_gid....... 00000000
sn_atime..... 00000000 02AB0F99 nsec
sn_mtime..... 00000000 02CB8368 nsec
sn_ctime..... 00000000 02CB8368 nsec
sn_acl....... 00000000

SN_VNODE.... 32ABD1DC
v_flag.... 00000000 v_count.... 00000018 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock@... 32ABD1E8 v_vfsp.... 01FB4000
v_mvfs... 00000000 v_gnode... 32B43080 v_next.... 00000000
v_vfsnext. 00000000 v_vfsprev. 00000000 v_pfsnode 14546080
v_audit... 00000000

SN_GNODE........ 32B43080
gn_type...... 00000009 gn_flags..... 00000000 gn_seg........ 007FF000
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt....... 00000000
(0)= more (^C to quit) ?
gn_wrcnt...... 00000000 gn_excnt..... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 32ABD1DC gn_rdev...... 00040000 gn_ops........ spec_vnops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 32B43084
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 32B43070
gn_type....... MPC

SN_PFSGNODE...... 145460C0
gn_type...... 00000004 gn_flags..... 00000000 gn_seg........ 00000000
gn_mwrcnt..... 00000000 gn_mrdcnt..... 00000000 gn_rdcnt....... 00000000
gn_wrcnt...... 00000000 gn_excnt..... 00000000 gn_rshcnt..... 00000000
gn_vnode...... 14546080 gn_rdev...... 00040000 gn_ops........ jfs_vops
gn_chan....... 00000000 gn_reclk_lock. 00000000 gn_reclk_lock@ 145460F4
gn_reclk_event FFFFFFFF gn_filocks.... 00000000 gn_data....... 14546080
gn_type....... CHR
KDB(0)>

Chapter 30. Display general and Journal File System (JFS) information subcommands   339
devnode subcommand

Purpose
The devnode subcommand displays device node table entries.

Syntax
devnode [slot | effectiveaddress]

Parameters

slot Specifies the slot number of a device node table entry. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of a device node table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary is displayed with one line per table entry. Display detailed information for individual devnode table entries by specifying either a slot number or an address.

Aliases
devno

eExample
The following is an example of how to use the devnode subcommand:

(0)> devnode //print device node table

DEVCNT SPECNODE GNODE LASTR PDATA TYPE

1 0B241758 00300000 1 0B2212E0 0B241768 00000000 05CB4E00 CHR
2 0B2211C8 00100000 1 00000000 0B2211C8 00000000 00000000 CHR
3 0B221140 00101000 1 0B221210 0B221150 00000000 00000000 BLK
4 0B221170 00020000 1 0B221170 0B221180 00000000 00000000 BLK
5 0B7E5A10 00120000 2 00000000 0B7E5A20 00000000 00000000 BLK
6 0B241070 00020000 1 0B8A3E0F 0B241080 00000000 00000000 BLK
7 0B2219AC 00200000 1 0B2219AC 0B2219BC 00000000 00000000 BLK
8 0B2219B8 00200000 1 0B2219C8 0B2219D8 00000000 00000000 BLK
9 0B7E5B00 00330000 1 0B7E5B00 0B7E5B10 00000000 00000000 BLK
10 0B221A10 00130000 1 0B221A10 0B221A20 00000000 00000000 BLK
11 0B2410B0 00330000 1 0B2410B0 0B2410C0 00000000 00000000 BLK
12 0B7E5A10 00130000 1 0B7E5A20 0B7E5A30 00000000 00000000 BLK
13 0B7E5C18 00330000 1 0B7E5C00 0B7E5C10 00000000 00000000 BLK
14 0B7E5B00 00130000 1 0B7E5B00 0B7E5B10 00000000 00000000 BLK
15 0B7E5A78 00330000 1 0B7E5A70 0B7E5A80 00000000 00000000 BLK
16 0B7E5C00 00330000 1 0B7E5C00 0B7E5C10 00000000 00000000 BLK
17 0B7E5C18 00330000 1 0B7E5C18 0B7E5C28 00000000 00000000 BLK
18 0B2416F0 00040000 1 0B2211A8 0B241700 00000000 00000000 BLK
19 0B221B80 00150000 3 0B221B80 0B221B90 00000000 05CB4E00 CHR
20 0B2410D0 00060000 1 0B221A80 0B2410E0 00000000 00000000 CHR

(0)> more (^C to quit) ? //Interrupted

(0)> devno 3 //print device node slot 3

DEVCNT SPECNODE GNODE LASTR PDATA TYPE

3 0B221940 00110000 2 00000000 0B221950 00000000 00000000 BLK

forw...... 00000000 back...... 00000000 lock...... 00000000

GNODE........... 0B221950
gn_type........ 00000003 gn_flags...... 00000000 gn_seg........ 007FFFFFF

340 KDB Kernel debugger and kdb command
Chapter 30. Display general and Journal File System (JFS) information subcommands

SPECNODES....... 00000000

...
fifonode subcommand

Purpose
The fifonode subcommand displays fifo node table entries

Syntax
fifonode [slot | effectiveaddress]

Parameters

slot Specifies the slot number of a fifo node table entry. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of a fifo node table entry. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary with one line per entry is displayed. Display detailed information for individual entries with either a slot number or an address.

Aliases
fifono

Example
The following is an example of how to use the fifono alias for the fifonode subcommand:

(0)> fifonode //print fifo node table

PFSGNODE SPECNODE SIZE RCNT WCNT TYPE FLAG

1 056D1C08 09D15EC8 082210D8 00000000 1 1 FIFO WWRT
2 056D1C08 09D15EC8 082210D8 00000000 1 1 FIFO RBLK WWRT

(0)> fifonode 1 //print fifo node slot 1

PFSGNODE SPECNODE SIZE RCNT WCNT TYPE FLAG

1 056D1C08 09D15EC8 082210D8 00000000 1 1 FIFO WWRT

ff_forw.... 0000D644 ff_back.... 0000D644 ff_dev..... FFFFFFF
ff_poll.... 00000001 ff_rptr.... 00000000 ff_wptr.... 00000000
ff_revent.. FFFFFFFF ff_wevent.. FFFFFFFF ff_buf..... 056D1C34

SPECNODE........ 082210D8
sn_next...... 00000000 sn_count..... 00000001 sn_lock...... 00000000
sn_gnode..... 082210E8 sn_pfsgnode.. 09D15EC8 sn_attr..... 00000000
sn_dev....... FFFFFFFF sn_chan....... 00000000 sn_vnode.... 056CE070
sn_ops...... 002731B0 sn_devnode.. 056D1C08 sn_type..... FIFO

SN_VNODE........ 056CE070
v_flag.... 00000000 v_count...... 000000002 v_vfsgen.. 00000000
v_lock.... 00000000 v_lock0.... 056CE07C v_vfsp.... 01AC9810
v_mvfsf.... 00000000 v_gnode... 082210E8 v_next.... 00000000
v_vfsnext.. 00000000 v_vfsprev. 00000000 v_pfsvnode 09D15EBB
v_aud... 00000000

SN_GNODE........ 082210E8
gn_type..... 00000008 gn_flags...... 00000008 gn_seg........ 007FFFFFF
gn_mwrcnt...... 00000000 gn_mrdcnt...... 00000000 gn_rdcnt...... 00000000
gn_wrcnt...... 00000000 gn_excnt...... 00000000 gn_rshcnt...... 00000000
gn_vnode...... 056CE070 gn_rdev...... FFFFFFFF gn_ops...... fifo_vnops
gn_chan........ 00000000 gn_recl_lock. 00000000 gn_recl_lock@ 0822111C
gn_recl_event 00000000 gn_filocks.... 00000000 gn_data...... 082210D8
gn_type..... FIFO
Chapter 30. Display general and Journal File System (JFS) information subcommands
hnode subcommand

Purpose
The hnode subcommand displays hash node table entries.

Syntax
hnode [bucket | effectiveaddress]

Parameters
bucket Specifies the bucket number within the hash node table. This parameter must be a decimal value.
effectiveaddress Specifies the effective address of a bucket in the hash node table. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is entered, a summary that contains one line per hash bucket is displayed. Display the entries for a specific bucket by specifying the bucket number or the address of the bucket.

Aliases
hno

Example
The following is an example of how to use the hno alias for the hnode subcommand:

(0)> hno //print hash node table

hnode+000000 1 0B241758  00000000  2
hnode+00000C 17 0B221940  00000000  1
hnode+00001C 26 05601C08  00000000  1
hnode+000018 33 0B221870  00000000  1
hnode+000018 34 0B7E5A10  00000000  2
hnode+000019 35 0B2219A8  00000000  1
hnode+000024 49 0B221808  00000000  1
hnode+000024 50 0B7E5B00  00000000  2
hnode+000025 51 0B241008  00000000  2
hnode+000026 52 0B7E5C18  00000000  2
hnode+000027 53 0B7E5A78  00000000  1
hnode+000027 54 0B7E5CB0  00000000  1
hnode+00002B 55 0B7E5CE8  00000000  1
hnode+000030 65 0B2416F0  00000000  1
hnode+00003C 61 0B221B00  00000000  1
hnode+000048 97 0B241008  00000000  1
hnode+000048 98 0B221848  00000000  1
hnode+000054 113 0B7E5AE0  00000000  1
hnode+000054 114 0B7E5EF0  00000000  1
hnode+000060 129 0B7E5B48  00000000  1

(0)> more (^C to quit) ? //Interrupted

(0)> hno 34 //print hash node bucket 34

HASH ENTRY ( 34): 00D0D0A4

<table>
<thead>
<tr>
<th>DEV</th>
<th>CNT</th>
<th>SPECNODE</th>
<th>GNODE</th>
<th>LASTR</th>
<th>PDATA</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0B7E5A10</td>
<td>00120001</td>
<td>2</td>
<td>00000000</td>
<td>0B7E5A20</td>
<td>00000000</td>
</tr>
<tr>
<td>2</td>
<td>0B241070</td>
<td>00020001</td>
<td>1</td>
<td>088A3EF0</td>
<td>0B241080</td>
<td>00000000</td>
</tr>
</tbody>
</table>

344 KDB Kernel debugger and kdb command
jfsnode subcommand

Purpose
The jfsnode subcommand prints details of the inode pool when no input parameter is provided. If the address of a jfs node is provided as an input parameter, the jfsnode subcommand verifies the jfs node and gives additional information on the related file system.

Note: This subcommand is only available in the kdb command.

Syntax
jfsnode [address]

Parameters
address Specifies the address of a node allocated in the inode cache.

Note: The address parameter is useful only for nodes allocated in the inode cache. It is not useful for soft mounts, specnodes, cdrnodes, or other non-jfs structures.

Aliases
jno

Example
The following is an example of how to use the jfsnode subcommand:

0)> jfsnode
INODES pool starts at 0x1101D6058
Static table[0] starts at 0xF100009E14793000, ends at 0xF100009E149C3000
Static table[1] starts at 0xF100009E149C3000, ends at 0xF100009E15053000
Static table[2] starts at 0xF100009E14BF3000, ends at 0xF100009E15283000
Static table[3] starts at 0xF100009E15283000, ends at 0xF100009E154B3000
Static table[4] starts at 0xF100009E156E3000, ends at 0xF100009E15913000
Static table[5] starts at 0xF100009E15913000, ends at 0xF100009E15B43000
Static table[6] starts at 0xF100009E15B43000, ends at 0xF100009E15D73000
Static table[7] starts at 0xF100009E15D73000, ends at 0xF100009E15FA3000
Static table[8] starts at 0xF100009E15FA3000, ends at 0xF100009E161D3000
Static table[9] starts at 0xF100009E161D3000, ends at 0xF100009E16403000
Static table[10] starts at 0xF100009E16403000, ends at 0xF100009E16633000
Static table[11] starts at 0xF100009E16633000, ends at 0xF100009E16863000
Static table[12] starts at 0xF100009E16863000, ends at 0xF100009E16A93000
Static table[13] starts at 0xF100009E16A93000, ends at 0xF100009E16CC3000
Static table[14] starts at 0xF100009E16CC3000, ends at 0xF100009E16EF3000
Static table[15] starts at 0xF100009E16EF3000, ends at 0xF100009E17123000
Static table[16] starts at 0xF100009E17123000, ends at 0xF100009E17353000
Static table[17] starts at 0xF100009E17353000, ends at 0xF100009E17583000
Static table[18] starts at 0xF100009E17583000, ends at 0xF100009E177B3000
Static table[19] starts at 0xF100009E177B3000, ends at 0xF100009E179E3000
Static table[20] starts at 0xF100009E179E3000, ends at 0xF100009E17C13000
Static table[21] starts at 0xF100009E17C13000, ends at 0xF100009E17E43000
Static table[22] starts at 0xF100009E17E43000, ends at 0xF100009E18073000
Static table[23] starts at 0xF100009E18073000, ends at 0xF100009E182A3000
Static table[24] starts at 0xF100009E182A3000, ends at 0xF100009E184D3000
Static table[25] starts at 0xF100009E184D3000, ends at 0xF100009E18703000
Static table[26] starts at 0xF100009E18703000, ends at 0xF100009E18933000
Static table[27] starts at 0xF100009E18933000, ends at 0xF100009E18B63000
Static table[28] starts at 0xF100009E18B63000, ends at 0xF100009E18D93000
Static table[29] starts at 0xF100009E18D93000, ends at 0xF100009E18F23000
Static table[30] starts at 0xF100009E18F23000, ends at 0xF100009E18F23000
Static table[31] starts at 0xF100009E18F23000, ends at 0xF100009E18F23000
Object (xnode) size is 0x230
vnode offset is 0x0
inode offset is 0x58
gnode offset is 0x78

(0)> jno 0xF100009E18B63000
0xF100009E18B63000 is a vnode

0xF100009E18B63000 is a vnode

Address not in jfs inode cache: 0x1

(0>
**kfset subcommand**

**Purpose**
The `kfset` subcommand displays the `kdm fset cache data` structure.

**Syntax**
`kfset address`

**Parameters**
`address` Identifies the address of the `kdm fset cache data` structure to display.

**Aliases**
kfs

**Example**
The following is an example of how to use the `kfset` subcommand:

```
KDB(0)> kfset 0x328A5400
linknxt.0x01FEB540  linkprv.0x01FEB540
fsid....0x000000000002C0007  refcnt..0x00000000002C0007
evdisp@.0x2FFBB394
k vnode.0x3173F1E0  fsetops.0x007FE300  attrnxt.0x328A5598  attrprv.0x328A5598
lock@.0x2FFBB520  options.0x00000000
mpath...0x3006A060  mplen...5  dpath...0x3006A080  dplen...12
attrnam.[ ]  class...0x00000000  subcls...0x00000000  length..0
```

Note: The kfset pointer is in the kdm vnode structure and may be obtained from the output of the kvnode command, in the fset field:

```
KDB(0)> kvnode 0x3173F1E0
enables..0x00000000  flags....0x00000000  nreg.....0x00000000
op.......0x0007FE320  fset....0x328A5400
regp.....0x00000000  data.....0x328389D8
```
Chapter 31. Display Enhanced Journaled File System
information subcommands

The subcommands in this category can be used to display Enhanced Journaled File System (JFS2) information. These subcommands include the following:

- `j2`
- `j2`
- `tree`
- `dt`
- `xl`
- `pgbuf`
- `pgobj`
- `txblock`
- `txblocki`
- `txlock`
- `bmblock`
- `j2no`
- `j2logbuf`
- `j2logx`
- `j2log`
- `pile`
- `slab`

© Copyright IBM Corp. 1997, 2006
j2 subcommand

**Purpose**
The j2 subcommand temporarily sets up access to the Enhanced Journaled File System (JFS2) metadata buffers so that the command specified as an input parameter can run correctly.

**Syntax**
```
j2 cmd
```

**Parameters**

- **cmd**: Indicates the actual kdb or KDB kernel debugger j2 subcommand that you want to run.

On the 32-bit kernel, there is a separate segment for JFS2 metadata. The j2 subcommand sets up the segment registers so that any command dealing with JFS2 metadata can examine the address in question as if JFS2 were attached.

On the 64-bit kernel, no address space setup is necessary, so the j2 subcommand runs the specified command.

The j2 subcommand is a wrapper that establishes the proper run environment for the specified subcommand that requires access to the j2 metadata.

**Aliases**

- jfs2

**Example**
The following is an example of how to use the j2 subcommand:
```
KDB(0)> j2 dd 0x00000000 20 //displays 32 words from the first page of the metadata segment
KDB(0)> j2 dd 0x00000000 20
D0000000: 4845415000043000 0000000000000001
            HEAP..0.........
D0000010: 0000FFBD00000000 0000000000000000
            ................
D0000020: 0000000000000000 0000000000000000
            ................
D0000030: 0000000000000000 0000000000000000
            ................
D0000040: 0000000000000000 0000000000000000
            ................
D0000050: 0000000000000000 0000000000000000
            ................
D0000060: 0000000000000000 0000000000000000
            ................
D0000070: 0000000000000000 0000000000000000
            ................
D0000080: 0000000000000000 0000000000000000
            ................
D0000090: 0000000000000000 0000000000000000
            ................
D00000A0: 0000000000000000 0000000000000000
            ................
D00000B0: 0000000000000000 0000000000000000
            ................
D00000C0: 0000000000000000 0000000000000000
            ................
D00000D0: 0000000000000000 0000000000000000
            ................
D00000E0: 0000000000000000 0000000000000000
            ................
D00000F0: 0000000000000000 0000000000000000
            ................
KDB(0)>
```

---

KDB Kernel debugger and kdb command
i2 subcommand

Purpose
The i2 subcommand displays the Enhanced Journaled File System (JFS2) inode.

Syntax
i2 [address | -c]
i2 [-d device] [-i inumber] [-m count]

Parameters
- address Displays the JFS2 inode structure at the specified inode address.
- -c Displays the inode cache table.
- -d device Displays a list of inodes in the specified device.
- -i inumber Displays the inode structure of the inode number specified.
- -m count Displays a list of inodes with a minimum number of the open count specified.

The -d, -i, and -m flags can be mixed. For these three flags, when multiple inodes satisfy the criteria, only summary information is displayed. If a single inode satisfies the criteria, detailed information is also displayed.

When the i2 command is invoked without any parameters, a summary list of inodes in memory is displayed along with the inodes' address, device, and inode number.

Aliases
inode2

Example
The following is an example of how to use the i2 subcommand:

KDB(0)> i2
ADDRESS  DEVICE  I_NUM  IPMNT  COUNT  TYPE  FLAG
325A0B00  000A000B  2    3252F080  00001  VDIR
32573080  000A000B  2    3252F080  00001  NON
325840B0  000A000B  0    00000000  00001  NON
3252F080  000A000B  mounted 3252F080  00001  NON
3252F080  000A000B  6    3252F080  00000  VDIR  CNEW
325D1080  000A000B  4    3252F080  00000  VREG  UPDNEW
325C1080  000A000B  5    3252F080  00000  VREG  UPDNEW
32595080  000A000B  16   3252F080  00000  VREG  UPDNEW
32584400  000A000B  35   3252F080  00000  VREG  UPDNEW
32534000  000A000B  34   3252F080  00000  VREG  UPDNEW
32563400  000A000B  33   3252F080  00000  VREG  UPDNEW
3252F080  000A000B  32   3252F080  00000  VREG  UPDNEW
3252F080  000A000B  64   3252F080  00000  VDIR  CNEW

KDB(0)> i2
ADDRESS  DEVICE  I_NUM  IPMNT  COUNT  TYPE  FLAG
325C1080  000A000B  4    3252F080  00000  VREG  UPDNEW

In-memory Working Inode:
hashCode....0x000002AF cacheClass...0x00000007 count.........0x00000000
capability...0x000001B7 atlhead........0x00000000 atttail......0x00000000
bxflag.......0x00000000 blid...........0x00000000 btindex........0x00000000
diocnt......0x00000000 nondiocnt....0x00000000
dev..........0x00000000 synctime.....0x403CE9A8 nodelock.....0x00000000
ipmnt.......0x3252F080 ipimap.......0x32595080 pagerObject..0x31A6D000

Chapter 31. Display Enhanced Journaled File System information subcommands
event......0xFFFFFFFF  fsevent......0xFFFFFFFF  openevent....0xFFFFFFFF
cacheList.nxt.0x317230B0  cacheList.prev.0x317230B0  freeNext.....0x317230B0
hashList.nxt..0x00000000  hashList.prev..0x31BA1034  kdmvp........0x00000000
flag........0x00000000
cflag........0x00000000
xlock........0x00000000
fsxlock......0x00000000
btorder......0x00000000
agstart......0x0000000000000000
lastCommittedSize...0x0000000000000000

Pseudo pagerBuffer @ 0x325C1124:
(0)> more (^C to quit) ?
tree subcommand

Purpose
The tree subcommand displays either the Enhanced Journaled File System (JFS2) d-tree or x-tree structure based on the specified inode parameter.

Syntax
tree address

Parameters
address Specifies the address of an inode. If the address of the specified inode is a directory, the d-tree structure is displayed. If the address of the specified inode is not a directory, the x-tree structure is displayed. This is a required parameter.

Aliases
No aliases.

Example
The following is an example of how to use the tree subcommand:

KDB(0) > tree 325C1080
flag........0x83
flag_name...BT_ROOT BT_LEAF
nextIndex....3
maxentry.....18
self.len.....0
self.addr1...0x00
self.addr2...0x00000000
self.addr...0
next.........0x34E0
prev.........0x34E0

Leaf xads:
xad[2]
flag........0x00
len..........1
addr1........0x00
addr2........0x00000028
off1..........0x00
off2........0x00000000
offset.......0
address.....40

xtree: Press [s]elect or e[x]it >
**dtree subcommand**

**Purpose**
The *dtree* subcommand displays the Enhanced Journaled File System (JFS2) d-tree structure and allows the user to walk the *dtree* structure.

**Syntax**
```
dtree address
```

**Parameters**

*address* Specifies the address of the d-tree structure.

The *dtree* subcommand contains its own subcommands that allow the user to walk the d-tree.

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Walks freelist entries.</td>
</tr>
<tr>
<td>s</td>
<td>Displays the specified slot entry.</td>
</tr>
<tr>
<td>t</td>
<td>Displays the formatted stbl structure.</td>
</tr>
<tr>
<td>u</td>
<td>Visits the parent node (but not the parent directory).</td>
</tr>
<tr>
<td>c</td>
<td>Visits the child node.</td>
</tr>
<tr>
<td>x</td>
<td>Exits subcommand mode.</td>
</tr>
</tbody>
</table>

**Aliases**
*dt*

**Example**
The following is an example of how to use the *dt* alias for the *dtree* subcommand:
```
KDB(0)> dt 0x325E1248
Internal D-tree page:
flag.........0x85
flag_name....BT_ROOT BT_INTERNAL
freecntr......7
Actual Free Count: 7
nextindex....1
freelist.....2
self.len.....0x010203
maxslot......0
stbblindex...0
self.addr1...0x04
self.addr2...0x85060708
next.........0x2
prev.........0x0

dtree: [n]ext, [f]reelist, [s]lot, s[t]bl, or e[x]it >
```
xtree subcommand

Purpose
The xtree subcommand displays the Enhanced Journaled File System (JFS2) xtree structure and allows the user to walk the xtree structure.

Syntax
xtree address

Parameters
address Displays the x-tree at the address of the specified x-tree.

The dtree subcommand contains its own subcommands that allow the user to walk the x-tree structure.

Subcommand Function
s Selects the xad entry to view.
u Visits the parent node.
c Visits the child node.
x Exits subcommand mode.

Aliases
xt

Example
The following is an example of how to use the xtree subcommand:

KDB(0)> xtree 0x325C1248
flag.........0x83
flag_name....BT_ROOT BT_LEAF
nextindex....3
maxentry.....18
self.len.....0
self.addr1...0x00
self.addr2...0x00000000
self.addr....0
next.........0x34E0
prev.........0x34E0

Leaf xads:
oxad[2]
flag........0x00
len..........1
addr1.......0x00
addr2.......0x00000028
off1........0x00
off2........0x00000000
offset......0
address......40

xtree: Press [s]elect or e[x]it >
pgobj subcommand

Purpose
The pgobj subcommand displays the Enhanced Journaled File System (JFS2) pager object structure.

Syntax
pgobj address

Parameters
address Displays the pager object structure at the specified address.

Aliases
No aliases.

Example
The following is an example of how to use the pgobj subcommand:

KDB(0)> pgobj 0x325B9000
flags........0x00000000 mCount.......0x00000000 cacheClass...0xFFFFFFFF
fileObject...0x325E1080 pageList.....0x325405C4 freeNext....0x31C8F000
pagerDevice..0x31C8F000 lock.........0x00000000 ioWait.......0xFFFFFFFF
deleteWait..0xFFFFFFFF xWait........0xFFFFFFFF mWaitShared..0xFFFFFFFF
mWaitExcl....0xFFFFFFFF pLastRead....0x0000000000000000 pTripWire....0x0000000000000000
l2LastReadAhead........0x00 12LastLastReadAhead......0x00
po_randReadTrust.....0x00000000 nPageLock...........0x00000000
cWriteBehind.0x0000000000000000 nRandomWrite........0x00000000

RBNA:
rbnaXoffset..0x0000000000000000 rbnaXlen..0x00000000
rbnaDelta.... 0x00 nRbnaXad..0xFFFFFFFF

wipXAD:
flag.........0x00000000 len..........0x00000000 addr1........0x00000000 addr2........0x00000000
off1........0x00000000 off2.........0x00000000
offset.......0x0000000000000000 address......0x0000000000000000

[]list pagerBuffer page list, e[x]it >
pgbuf subcommand

Purpose
The pgbuf subcommand displays the Enhanced Journaled File System (JFS2) pager buffer structure.

Syntax
pgbuf address | -c

Parameters
address Displays the JFS2 pager buffer structure at the specified address.
-c Displays a list of the JFS2 pager buffers in the buffer cache.

Aliases
No aliases.

Example
The following is an example of how to use the pgbuf subcommand:

KDB(0)> pgbuf -c
jCache:
nBuffer: 0xA00 (2560)
nCacheClass: 9
minFreePerCC: 5
maxFreePerCC: 8
nHashClass: 0x3FF (1023)
cacheTable: 0x31A70000
hashTable: 0x31C0E000
freeWait: 0xFFFFFFFF
jCacheClassLow: 0

Cache table:
<table>
<thead>
<tr>
<th>CLASS</th>
<th>BUFS</th>
<th>FREE</th>
<th>LRU</th>
<th>CACHELIST.HEAD</th>
<th>FREELIST.HEAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3253F8DC</td>
<td>3253F90</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>32540214</td>
<td>3253F17C</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3253F268</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3253F354</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3253F440</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3253F52C</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3253FE64</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3253FF50</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3253F7F0</td>
<td>0</td>
</tr>
</tbody>
</table>

KDB(0)> pgbuf 3253F8DC
xflag........0x0000000C  BUFFER PAGE
nohomeok........0x00000000
lid...........0x00000000
flags.........0x00000011  METADATA IODONE
count............0x00000000  cacheClass........0x00000000
data............0x00000000  syncList.nxt.0x00000000  syncList.prv.0x00000000
log............0x00000000  ip................0x32595080  pagerObject..0x325A7000
pagelist.nxt.0x00000000  pagelist.prv.0x3253FA08  hashlist.nxt.0x00000000
hashlist.prv.0x31C10460  cachelist.nxt.0x32540128  cachelist.prv.0x31A70010
ioListNext....0x32540128  freeList.nxt.0x32540128  freeList.prv.0x31A70010
ioNext.........0x00000000  iobp..........0x3253F884  ioWait........0xFFFFFFFF
waitList.......0xFFFFFFFF
lsn...........0x0000000000000000  clsn........0x0000000000000000
xoffset........0x0000000000000000  pxd........0x000000000000001E

KDB(0)>
txblock subcommand

Purpose
The txblock subcommand displays the Enhanced Journaled File System (JFS2) transaction block structure.

Syntax

```
txblock address
```

Parameters

`address` Displays the transaction block at the specified address.

Aliases

`txblk`

Example

The following is an example of how to use the `txblock` subcommand:

```
KDB(3)> txblock 32503108
xflag........0x00000000  flag........0x00000000  next........0x00000000
locker........0x00000000  eor...........0x00000000  logTid........0x00000005
lidList......0x2FF3ABA8  waitor.......0xFFFFFFFF  lwbmp........0x00000000
bp...........0x00000000  cqnext.......0x00000000  gcWait.......0xFFFFFFFF
ipmnt........0x325C1780  lwmlsn.......0x00000000  clsn.........0x00000000
lspn.........0x00000000  lwm1sn.......0x00000000  lspn.........0x00000000
KDB(3)> 
```
 txblocki subcommand

Purpose
The txblocki subcommand displays the Enhanced Journaled File System (JFS2) transaction block.

Syntax
txblocki index

Parameters
index Displays the transaction block at the specified index.

Aliases
txbli

Example
The following is an example of how to use the txblocki subcommand:

KDB(0)> txblocki 1
xflag........0x00000000 flag........0x00000000 next........0x00000000
locker........0x00000000 eor...........0x00000000 logTid........0x00000000
lidList......0x2F3ABA8 waitor........0xFFFFFFFF lwmbp........0x325411C0
bp...........0x00000000 cqnext........0x00000000 gcWait........0xFFFFFFFF
ipmnt........0x325C1780 lwmsn........0x0000000000006F3B
clsn..........0x0000000000000000 lspn.........0x0000000000000000
KDB(3)>
txlock subcommand

Purpose
The txlock subcommand displays the Enhanced Journaled File System (JFS2) transaction lock structure.

Syntax
```
taxlock address
```

Parameters

- `address`: Displays the transaction lock structure at the specified address.

Aliases
- txlck
- txick

Example
The following is an example of how to use the `txlock` subcommand:

```
KDB(3)> txlock 2FF3ABA8
  tid...........0x00000003
  flag.........0x00008801 PAGELOCK LOG LOCAL
  next.........0x2FF3AB60 ip.........0x32573B00
  bp...........0x325411C0 lock........0x00000000
  type.........0x00008002 GROW ENTRY INODE

  maxcnt.......0x00000016  l2linesize...0x00000004
  index........0x00000001

  lv[0].offset.0x00000040 lv[0].length.0x00000008
  next.........0x00000000

KDB(3)>
```
bmblock subcommand

Purpose
The bmblock subcommand displays the Enhanced Journaled File System (JFS2) metadata block and tries to lookup the hash value for a particular block and see if it exists in the cache.

Syntax
bmblock ipAddr xoff block | page | raw

Parameters
ipAddr    Specifies the address of an inode.
xoff      Specifies the offset.
block, page, raw  Specifies the page buffer type. One of these values must be provided.

Aliases
bmb, bmblk

Example
The following is an example of how to use the bmb alias for the bmblock subcommand:

    (0)> bmb 0xF10010F00F655C80 1C72F block
    Hashclass @ F10010F00F495700
    Pager buffer @ F10010F00F73C128
    xflag........0x00000000 BUFFER BLOCK
    nohomeok.....0x00000000
    lid..........0x0000000000000000
    flags........0x00020011 METADATA IODONE HIT
    count...........0x00000000 cacheClass...........0x00000002
    data.........0xF10010A11006E000 logx...........0x0000000000000000 syncList.nxt.0xF10010F00F31DE8
    syncList.prv.0xF10010F00F73C330 hashList.nxt.0xF10010F00F495700
    hashList.prv.0xF10010F00F8F22E8 cacheList.nxt.0xF10010F00F107EE058
    cacheList.prv.0xF10010F00F8F22E8 freeList.nxt.0xF10010F00F107EE058
    freeList.prv.0xF10010F00F8F22E8 iobp...........0xF10010F00F73C058
    iobp.0x0000000000000000
    waitList.....0xFFFFFFFFFFFFFFFF
    lsn........0x0000000000000000
    xoffset......0x000000000001C72F
    pxd...........0x000000000001C72F
jfs2node subcommand

Purpose
The jfs2node subcommand displays the Enhanced Journaled File System (JFS2) xnode structures.

Syntax
jfs2node address

Parameters
address Specifies an address at which to check whether that address is a valid JFS2 xnode structure or an offset into one. If there is a valid xnode or offset, the jfs2node subcommand displays the relevant structure. This is a required parameter.

Aliases
j2no

Example
The following is an example of how to use the j2no alias for the jfs2node subcommand:

(0)> j2no 0x1
0x1 is not a valid JFS2 xnode address.

(0)> i2
ADDRESS DEVICE I_NUM IPMNT COUNT TYPE FLAG
369F9080 00220001 1 369C9080 00001 NON
369C9080 00220001 mounted 369C9080 00001 NON
36A1F080 00220002 0 00000000 00001 NON
36A40308 00220001 2 369C9080 00001 VDIR
36A0C080 00220001 2 369C9080 00001 NON
36A30080 00220001 16 369C9080 00001 NON

(0)> j2no 36A1F080
0x36A1F080 is an inode:

In-memory Working Inode:
hashClass...0x00000422 cacheClass...0x00000005 count.......0x00000001
capability...0x00000125 atlhead......0x00000000 atltail......0x00000000
bxflag......0x00000000 blid.........0x00000000 btindex......0x00000000
diocnt......0x00000000 nondiocnt....0x00000000
dev..........0x00000000 synctime.....0x00000000 nodelock.....0x00000000
ipmnt........0x00000000 ipimap.......0x00000000 pagerObject..0x00000000
event.........0xFFFFFFFF fsevent.....0xFFFFFFFF openevent....0xFFFFFFFF
cacheLst.nxt.0x00000000 cacheLst.prv.0x00000000 freeNext....0x00000000
hashLst.nxt.0x00000000 hashLst.prv..0x366BE198 kdmvp........0x00000000
flag.........0x00000100 flag_type....SYSTEM
cflag........0x00000000 xlock........0x00000000
fsxlock.......0x00000000 btxlock.....0x00000000
btorder......0x00000000
agstart......0x0000000000000000 lastCommittedSize...0x0000000000000000
.
.
.

(0)> j2no 0x36A1F085
0x36A1F085 is at offset 5 into wInode:
In-memory Working Inode:
hashClass...0x00000422  cacheClass...0x00000005  count.......0x00000001
capability...0x00000125  atlhead.....0x00000000  atltail......0x00000000
bxflag.......0x00000000  blid.........0x00000000  btindex......0x00000000
diocnt......0x00000000  nondiocnt....0x00000000

.<as above>

.

.
j2logbuf subcommand

Purpose
The **j2logbuf** displays the Enhanced Journaled File System (JFS2) log buffer structure.

Syntax
```
j2logbuf address
```

Parameters

- **address** Displays the JFS2 log buffer at the specified address.

Aliases
No aliases.

Example
The following is an example of how to use the **j2logbuf** subcommand:

```
KDB(0)> j2logbuf 31D6F5C4
lb_flags..........0x0000210C  LB_WRITE LB_GC LB_IOCONE LB_IOERROR
lb_lhs...........0x00000000000001246  lb_cls...........0x0000000000000000
lb_ceor..........0x000000378  lb_bkno..........0x0000000000000000
lb_cn...........0x0000001246  lb_eor..........0x000000378
syncList.nxt....0x00000000  syncList.prv....0x00000000
pageList.nxt....0x00000000  pageList.prv....0x00000000
hashList.nxt....0x00000000  hashList.prv....0x00000000
cacheLst.nxt....0x00000000  cacheLst.prv....0x00000000
freeList.........0x00000000  iobNext...........0x31D6F5C4
waitList.........0xFFFFF7FF  data...........0xD005A000
iowait...........0xFFFFF7FF  iobp...........0x31D6F5C4
KDB(0)>
```
**j2logx subcommand**

**Purpose**
The `j2logx` subcommand displays the logx structure.

**Syntax**

```
j2logx [address]
```

**Parameters**

`address` Displays the logx structure at the specified address.

**Aliases**
No aliases.

**Example**

The following is an example of how to use the `j2logx` subcommand:

```
KDB(0)> j2logx 31D6C000
flag.........0x00000000 count........0x00001E02 errCount.....0x00000204
hwmErrCount..0x00001000 lwmErrCount..0x00000020
lsn..........0x00000001246378 clsn.........0x00000000000015BE
size..........0x000000002000000 space........0x01FFE000
syncpt........0x0000000010C23B0 sync........0x0000000010C23B0
nFreeBuffer....0x00000002 nBuffer........0x00000000
hwmBuffer......0x000000280 lwmBuffer.......0x000000140
pageOutQueue...0x31D6F5C4 freeBufferList..0x31C03440
lwmBufferWait...0xFFFFFFFF freeBufferWait..0xFFFFFFFF
syncListLock...0x00000000 ioLock.........0x00000000
syncList.head...0x3283A9B8 syncList.tail...0x32833B4C
freeList........0x00000000 iLogSyncCursor.@.0x005A7198
bmLogSyncCursor.@.0x005A71C0 bmLogSyncRCursor.@.0x005A71E8
KDB(0)>
```
j2log subcommand

Purpose
The j2log subcommand displays the log-t structure.

Syntax
j2log address

Parameters
address Specifies a valid address for the log-t structure.

Aliases
No aliases.

Example
The following is an example of how to use the j2log subcommand:

KDB(0) > j2log 32557400
di_number......0x0000000000000000
di_gen.........0x00000000 di_filesit.....0x00000000
serial.........0x00000000000000002C base........0x0000000000000000
flag........0x00000100
state........0x00000000 LOGIOERROR
size.........0x00000000 bsize........0x00000000
pbsize.......0x00000000 12bsize.....0x0000 12pbsize.....0x0000
logTid.......0x000000001246
pn........0x0000000000000000 cntr........0x0000000000000000
syncState....0x00000000 nextsync.....0x001FFFF0
active.......0x00000000 syncBarrier.....0x00000000 syncTid.......0x00000000
after wnode, start at 0x005A70F8
bp........0x31D6F5C4 dev........0x000A00A
devfp........0x100038A0 logx........0x31D6C00
logList.nxt..0x00000000 logList.prv..0x00806F7C
rdwrLock.....0x00000000 logLock......0x00000000
CMQ.head.....0x00000000 CMQ.tail.....0x00000000
gclrt........0x324F30B0 gcLock.......0x00000000
syncWait.....0x00000000 nTxLog........0x00000000
KDB(0) >
pile subcommand

Purpose
The pile subcommand displays information about pile data structures.

Syntax
pile [address]

Parameters
address Specifies the memory address of the pile structure.

The pile subcommand can be run in the following ways:
• If no argument is specified, the pile subcommand lists the addresses of all the piles on the system and validates the pile identifier of the specified pile.
• If an argument is specified, the pile subcommand attempts to print the contents of that address as a pile structure and validates the pile ID of every pile in the system.

If a valid pile identifier is not detected, an error message is displayed.

Aliases
No aliases.

Example
The following is an example of how to use the pile subcommand:

KDB(0)> pile

ADDRESS  NAME     cur_total_pages
0x3004B380 NLC64     0x0000000000000004
0x3004B400 NLC128    0x0000000000000000
0x3004B480 NLC256    0x0000000000000000
0x328A6E600 iCache   0x0000000000000010
0x328A6E580 iCache   0x0000000000000010
0x328A6E480 iCache   0x0000000000000010
0x328A6E500 iCache   0x0000000000000010
0x328A6E300 iCache   0x0000000000000010
0x328A6E380 bmIOBufPile 0x0000000000000000
0x328A6E680 bmXBufPile 0x0000000000000004
0x328AE700 j2VBufferPool 0x0000000000000000
0x328AE780 j2VBufferPool 0x0000000000000000
0x328AE800 j2VBufferPool 0x0000000000000000
0x328AE880 j2VBufferPool 0x0000000000000028
0x328AE900 j2VBufferPool 0x0000000000000000
0x328AE980 dioCache   0x0000000000000004
0x328AE980 dioCache   0x0000000000000000
0x328AE900 dioCache   0x0000000000000000
0x328AE980 dioCache   0x0000000000000000
KDB(0)> pile 0x3004B380

name........NLC64
prev........0x328AE800 next........0x3004B400
ID..........0x50494C45 objectsize..0x0044 align.......0x0003
slabsize....0x0004 intpri.....0x000B flags.......0x00000000
maxtotalpg..0xFFFFFFFFFFFFFFFF mintotalpg..0x0000000000000000
curtotalpg..0x0000000000000000 slab_full..0x3004B3A8 squeezed.....0x0000 full........0x0000
slab_part..0x32D4D000 partial....0x0001 empty.......0x0000
slab_dead..0x0 dead.........0x0000
pile_lock..0x00000000 alloc_lock..0x00000000
heap........0x300000B8
HANDLERS:
  cookie.......0x00000000  reconfig....0x00000000
  init........0x00000000  free.........0x00000000

KDB(0)>
slab subcommand

Purpose
The slab subcommand displays the slab structure at the specified address.

Syntax
slab address

Parameters
address Specifies the memory address for which you want to display the slab structure. The address parameter is required.

The slab command performs some basic error checking on the data structure. If the slab subcommand finds an invalid slab ID, a warning message is generated. If the pile to which the slab belongs has an invalid ID, a warning message is generated.

Aliases
No aliases.

Example
The following is an example of how to use the slab subcommand:
KDB(0)> slab 0x337EC000
Pile........0x32BAE600
ID.........0x534C4142 prev........0x32BAE630 next........0x32BAE630
freelist....0x337EC3FC datastart...0x337EC07C objsize.....0x0380
flags.......0x0005 refcount....0x00000001 maxrefcnt...0x00000049
pages.......0x0010 pagesinuse..0x0010

KDB(0)>
Chapter 32. Display NFS information subcommands

The subcommands in this category can be used to display NFS information. These subcommands include the following:

- `cupboard`
- `sockpint`
- `sockcup`
- `svcxeprt`
cupboard subcommand

Purpose
The `cupboard` subcommand displays either a list of the current KRPC server cupboards in use or displays the contents of a single KRPC server `cupboard` structure.

Syntax
`cupboard [effectiveaddress]`

Parameters
`effectiveaddress` Specifies the effective address of a `cupboard` structure to display. If this parameter is omitted, a list of the current KRPC server cupboards is displayed.

Aliases
No aliases.

Example
The following is an example of how to use the `cupboard` subcommand:

```
KDB(0)> cupboard 3286BE00 rpc.lockd
KDB(0)> cupboard 3286BE00
CUPBOARD............ 3286BE00
RPC Services:
program 100021, Version 4, Dispatch .lm_nlm4_dispatch
program 100021, Version 3, Dispatch .lm_nlm_dispatch
program 100021, Version 2, Dispatch .lm_nlm_dispatch
program 100021, Version 1, Dispatch .lm_nlm_dispatch

Service Handles:
Address   Sockpoint
32D4BD00   3286BE00 Master UDP handle - receiving on port 32769
3285D100   3286BE00 Master UDP handle - receiving on port 32788

Manager Section:
cb_mgrlock...... 00000000  cb_event........ FFFFFFFF
  cb_all_stop..... FALSE  cb_wrap.......... FALSE
  cb_start_thread. FALSE
  cb_mgr_thread... 00004D9F  cb_mon_thread... 0000429F
  cb_svc_thread... 00004B9D  cb_ogre_thread.. 00004EA1
  cb_xprt.......... 32D4BD00  cb_free_xprt.... 32D4B800
  cb_next.......... 00000000  cb_name......... rpc.lockd

Count Section:
cb_maxthreads. 00000020  cb_threads. 00000005
  cb_active...... 00000000  cb_head...... 00000001
  cb_idle........ 00050000  cb_idle5...... 00050000
  cb_idle15...... 0004FC00  cb_reserve.... 00000000
  cb_threads1... 00050000  cb_threads5. 00050000
  cb_threads15.. 0004FC08

Sockcup Section:
cb_sclock....... 00000000  cb_scfree......... 32BE2780
  cb_scfirst...... 00000000  cb_sclast........... 00000000
  cb_num_sockcups.000005C  cb_queued_sockcups.. 00000000
  cb_queued1..... 00000000  cb_ququked5......... 00000000
  cb_queued15.... 0000013B

Service Threads Waiting:
```
<table>
<thead>
<tr>
<th>Thread Slot</th>
<th>Service Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>32D5D300</td>
</tr>
<tr>
<td>64</td>
<td>32D4BF00</td>
</tr>
<tr>
<td>72</td>
<td>32D4B900</td>
</tr>
<tr>
<td>68</td>
<td>32D4BE00</td>
</tr>
<tr>
<td>75</td>
<td>32D4B700</td>
</tr>
</tbody>
</table>

Queued Sockcups:
None

KDB(0)>
sockpint subcommand

Purpose
The `sockpint` subcommand displays the contents of a KRPC server `sockpint` structure.

Syntax
`sockpint effectiveaddress`

Parameters
`effectiveaddress`  Specifies the effective address of the `sockpint` structure to display.

Aliases
No aliases.

Example
The following is an example of how to use the `sockpint` subcommand:

```
KDB(0)> sockpint 34FFA8C
SOCKPINT............ 0034FFA8C
sp_lock... 0194387B  sp_expand_lock. 12F05400  sp_event.... 20363AF8
sp_xprrt... 00067C00  sp_cupboard.... F8505400  sp_socket... 38307EC0
sp_ref.... 1A144800  sp_time........ 00018063
sp_queued... B42B3800
KDB(0)>
```
**sockcup subcommand**

**Purpose**
The *sockcup* subcommand displays the contents of a KRPC server *sockcup* structure.

**Syntax**
*sockcup effectiveaddress*

**Parameters**

*effectiveaddress* Specifies the effective address of the *sockcup* structure to display.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the *sockcup* subcommand:

```
KDB(0)> sockcup 3D32532
SOCKCUP.......... 003D32532
Next.. 0194387B Mbuf.. 12F05400 Sockpint.. 20363AF8
KDB(0)> 
```
svcxprt subcommand

Purpose
The svcxprt subcommand displays the contents of a KRPC server svcxprt structure.

Syntax
svcxprt effectiveaddress

Parameters

effectiveaddress  Specifies the effective address of the svcxprt structure to display.

Aliases
No aliases.

Example
The following is an example of how to use the svcxprt subcommand:

KDB(0)> svcxprt 428C82
SVCXPR............ 00428C82
xp_next.... 0194387B xp_tid....... 12F05400 xp_flags...... 20363AF8
xp_cb......... 00003800 xp_sp......... 000C3BA0 xp_sock........ 00067C00
xp_ops........ 38307EC0 xp_cred....... 1A144800 xp_type....... B42B3800
xp_sockout.... 00000CB0 xp_socksendsz. 00003878 xp_sockrecvsz. 0F5462C4
xp_p1.......... 0020A063 xp_p2.......... 00182803 xp_p3.......... 00014181
xp_read_dsb.... 000038A0 xp_closeproc.. 00084BCF xp_callouts... 64558375
xp_maxthreads. 0001B005 xp_minthreads. 00064BCC xp_addrlen.... 00018063
xp_port........ F850
xp_sockcup..... 000C48F6 93E56060 00009061
xp_verf........ 02146085 0000A084 00063804
KDB(0)>
Chapter 33. Time subcommands

The subcommands in this category are used to determine the elapsed time from the previous use of the KDB kernel debugger, and to determine Timer Request Block (TRB) information. These subcommands include the following:

- `time`
- `trb`
time subcommand

Purpose
The time subcommand determines the elapsed time from the last time the KDB kernel debugger was exited to the time it was entered.

Note: The time subcommand is only available in the KDB kernel debugger. It is not included in the kdb command.

Syntax
time

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the time subcommand:

```
KDB(4)> debug ? //debug help
vmm Hw lookup debug... on with arg 'dbg1++', off with arg 'dbg1--'
vmm tr/tv cmd debug... on with arg 'dbg2++', off with arg 'dbg2--'
vmm SW lookup debug... on with arg 'dbg3++', off with arg 'dbg3--'
symbol lookup debug... on with arg 'dbg4++', off with arg 'dbg4--'
stack trace debug..... on with arg 'dbg5++', off with arg 'dbg5--'
BRKPT debug (list).... on with arg 'dbg61++', off with arg 'dbg61--'
BRKPT debug (instr).... on with arg 'dbg62++', off with arg 'dbg62--'
BRKPT debug (suspend). on with arg 'dbg63++', off with arg 'dbg63--'
BRKPT debug (phantom). on with arg 'dbg64++', off with arg 'dbg64--'
BRKPT debug (context). on with arg 'dbg65++', off with arg 'dbg65--'
Dabr debug (address)... on with arg 'dbg71++', off with arg 'dbg71--'
Dabr debug (register). on with arg 'dbg72++', off with arg 'dbg72--'
Dabr debug (status)... on with arg 'dbg73++', off with arg 'dbg73--'
BRAT debug (address)... on with arg 'dbg81++', off with arg 'dbg81--'
BRAT debug (register). on with arg 'dbg82++', off with arg 'dbg82--'
BRAT debug (status)... on with arg 'dbg83++', off with arg 'dbg83--'
BRKPT debug (context). on //this debug feature is enable
KDB(4)> debug dbg5++ //enable debug mode
stack trace debug..... on
KDB(4)> f //stack frame in debug mode
thread+000180 STACK:
--- Look for traceback at 0x00015278
--- Got traceback at 0x00015280 (delta = 0x00000008)
--- has_tboff = 1, tb_off = 0x08
--- Trying to find Stack Update Code from 0x000151A8 to 0x00015278
--- Found 0x9421FFA0 at 0x000151B8
--- Trying to find Stack Restore Code from 0x000151A8 to 0x0001527C
--- Trying to find Registers Save Code from 0x000151A8 to 0x00015278
[00015278]waitproc+000000 ()
--- Look for traceback at 0x00015274
--- Got traceback at 0x00015280 (delta = 0x0000000C)
--- has_tboff = 1, tb_off = 0x08
[00015274]waitproc+0000CC ()
--- Look for traceback at 0x0002F400
--- Got traceback at 0x0002F420 (delta = 0x0000020)
--- has_tboff = 1, tb_off = 0x30
[0002F400]procentry+000010 (??, ??, ??, ??)
```
Invoke command from command line that calls open

Breakpoint

// Invoke command from command line that calls open
Breakpoint
0024FDE8 stwu stkp,FFFFFFB0(stkp) stkp=2FF3B3C0,FFFFFFB0(stkp)=2FF3B370
KDB(0)> time // Report time from leaving the debugger till the break
Command: time Aliases:
Elapsed time since last leaving the debugger:
2 seconds and 12121136 nanoseconds.
KDB(0)>
trb subcommand

Purpose
The trb subcommand displays Timer Request Block (TRB) information.

Syntax
trb [ * | cpu x ] [ option ]

Parameters
* Displays Timer Request Block (TRB) information for TRBs on all processors. Summary information is displayed for some options. To see detailed information, select a specific processor and option.

cpu x Is the text cpu followed by the processor number. It displays TRB information for the specified processor. 
**Note:** The characters cpu must be included in the input. The value x is the hexadecimal number of the processor.

option Specifies the option number that indicates the data to be displayed. The available option numbers can be viewed by entering the trb subcommand with no arguments.

If this subcommand is entered without parameters, a menu displays that allows you to select the data you want to display.

Aliases
timer

Example
The following is an example of how to use the trb subcommand:

```
KDB(4)> trb //timer request block subcommand usage
Usage: trb [CPU selector] [1-9]
CPU selector is ' * ' for all CPUs, 'cpu n ' for CPU n, default is current CPU

Timer Request Block Information Menu
  1. TRB Maintenance Structure - Routine Addresses
  2. System TRB
  3. Thread Specified TRB
  4. Current Thread TRB's
  5. Address Specified TRB
  6. Active TRB Chain
  7. Free TRB Chain
  8. Clock Interrupt Handler Information
  9. Current System Time - System Timer Constants
Please enter an option number: //<CR/LF>
```

```
KDB(4)> trb * 6 //print all active timer request blocks
```

```
CPU #0 Active List
  CPU PRI ID SECS NSECS DATA FUNC
05689080  0000 0005 FFFFFFFE 00003BBA 23C303B0 05689080 sys_timer+000000
05689080  0000 0003 FFFFFFFE 00003BBA 27DAC680 00000000 pfastsched+000000
05689580  0000 0003 FFFFFFFE 00003BBA 2911B080 00000000 pfslowscd+000000
0805A3C0  0000 0005 000001751 00003BBA 20D8C480 0805A3C0 rtsleep_end+000000
05689500  0000 0003 FFFFFFFE 00003BBA 23166800 00000000 if_slowscd+000000
0805A480  0000 0003 FFFFFFFE 00003B8F 20585280 00000000 018633F0

CPU #1 Active List
  CPU PRI ID SECS NSECS DATA FUNC
05689100  0001 0005 FFFFFFFE 00003BBA 23C38E80 05689100 sys_timer+000000
```

380  KDB Kernel debugger and kdb command
### CPU #2 Active List

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689180</td>
<td>0002</td>
<td>0005</td>
<td>FFFFFFFE</td>
<td>00003BBA</td>
<td>23C37380</td>
<td>05689180 sys_timer+000000</td>
</tr>
<tr>
<td>0805A500</td>
<td>0002</td>
<td>0005</td>
<td>00001525</td>
<td>00003BBAE</td>
<td>0CFF9500</td>
<td>0805A518 rtsleep_end+000000</td>
</tr>
</tbody>
</table>

### CPU #3 Active List

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689200</td>
<td>0003</td>
<td>0005</td>
<td>FFFFFFFE</td>
<td>00003BBA</td>
<td>23C39F80</td>
<td>05689200 sys_timer+000000</td>
</tr>
</tbody>
</table>

(4)> more ("C to quit")? //continue

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689880</td>
<td>0003</td>
<td>0005</td>
<td>00000003</td>
<td>00003BBA</td>
<td>01B73180</td>
<td>00000000 sched_timer_post+000000</td>
</tr>
<tr>
<td>0805A580</td>
<td>0003</td>
<td>0005</td>
<td>00000001</td>
<td>00003BBA</td>
<td>08CA7300</td>
<td>0000000E interval_end+000000</td>
</tr>
</tbody>
</table>

### CPU #4 Active List

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689280</td>
<td>0004</td>
<td>0005</td>
<td>FFFFFFFE</td>
<td>00003BBA</td>
<td>23C39800</td>
<td>05689280 sys_timer+000000</td>
</tr>
</tbody>
</table>

### CPU #5 Active List

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689300</td>
<td>0005</td>
<td>0005</td>
<td>FFFFFFFE</td>
<td>00003BBA</td>
<td>23C39800</td>
<td>05689300 sys_timer+000000</td>
</tr>
<tr>
<td>05689780</td>
<td>0005</td>
<td>0005</td>
<td>FFFFFFFF</td>
<td>00003BFF</td>
<td>1B052C00</td>
<td>01C62C40 01ADD6FC</td>
</tr>
</tbody>
</table>

### CPU #6 Active List

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689380</td>
<td>0006</td>
<td>0005</td>
<td>FFFFFFFE</td>
<td>00003BBA</td>
<td>23C3C200</td>
<td>05689380 sys_timer+000000</td>
</tr>
</tbody>
</table>

### CPU #7 Active List

<table>
<thead>
<tr>
<th>CPU</th>
<th>PRI</th>
<th>ID</th>
<th>SECS</th>
<th>NSECS</th>
<th>DATA</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>05689400</td>
<td>0007</td>
<td>0005</td>
<td>FFFFFFFE</td>
<td>00003BBA</td>
<td>23C38180</td>
<td>05689400 sys_timer+000000</td>
</tr>
<tr>
<td>05689680</td>
<td>0007</td>
<td>0003</td>
<td>FFFFFFFF</td>
<td>00003BFF</td>
<td>00000000</td>
<td>00000000 threadtimer+000000</td>
</tr>
</tbody>
</table>

KDB(4)> trb cpu 1 6 //print active list of processor 1

**CPU #1 TRB #1 on Active List**

- Timer address.................05689100
- trb->to_next......................00000000
- trb->knext......................00000000
- trb->kprev......................00000000
- Owner id (-1 for dev drv)........FFFFFFFE
- Owning processor..................00000001
- Timer flags......................00000013 PENDING ACTIVE INCINTERVAL
- trb->eventlist...................FFFFFFFE
- trb->timeout.it_interval.tv_sec...00000000
- trb->timeout.it_interval.tv_nsec..00000000
- Next scheduled Timeout (secs).....00003BBA
- Next scheduled timeout (nanosecs)...23C38E80
- Completion handler..............00083BA4 sys_timer+000000
- Completion handler data.........05689100
- Int. priority....................00000005
- Timeout function................00000000 00000000

KDB(4)}>

---

Chapter 33. Time subcommands 381
Chapter 34. System trace, dump and error log subcommands

The subcommands in this category support some fundamental AIX Reliability and Serviceability features. These subcommands display data in the kernel trace buffers, data in the trace buffers, unprocessed system error log entries, and data in a system memory dump. These subcommands include the following:

- `trace`
- `trcstart`
- `trcstop`
- `cdt`
- `errpt`
- `mtrace`
- `check`
- `ctctrl`
trace subcommand

Purpose
The **trace** subcommand displays data in the system trace buffers or data in the KDB kernel debugger trace buffers collected using the `trcstart subcommand` on page 386.

Syntax
```
trace [-h] [-c channel] [hook [:subhook]]... [ #data ]... [-t TID] [-v]
```
```
trace -K [-j event_list] [-k event_list] [-t TID] [-v]
```

Parameters
- **-h** Displays trace headers.
- **hook** Specifies the hexadecimal value of the hook IDs on which to report. Multiple hook IDs should be separated by spaces.
- **:subhook** Specifies subhooks, if needed. The subhooks are specified as hexadecimal values.
  **Note:** If subhooks are used, the complete syntax must include both the hook and subhook IDs as a pair separated by a colon. For example, assume the trace hook is 1d1 and the subhook is 2d. The hook specification would look like 1d1:2d. Separate multiple hook and subhook pairs with a space.
- **data** Selects only those trace entries with the specified data word. For instance, specifying #AA displays all trace entries where at least one data word is recorded as 0xAA.
- **-c channel** Selects the trace channel for which the contents are to be monitored. The value for channel must be a decimal constant in the range 0 to 7. If no channel is specified, a prompt is displayed.
- **-K** Displays the trace gathered using the `trcstart` subcommand.
- **-j event_list** Displays trace data only for the events in the list.
- **-k event_list** Displays trace data for the events that are not in the list.
- **-t TID** Thread ID filter. A thread is identified by its thread ID in hex format. When the **trace** command is used to display buffer contents, this parameter is used to display only events related to the specified thread. Otherwise, this parameter is not valid. This parameter can be used when both system and KDB kernel debugger trace buffers are displayed.
- **-v** Displays events using a verbose output format. This parameter can be used when both system and KDB kernel debugger trace buffers are displayed.

To display the user-initiated system trace buffers, specify the **channel** (**-c**). Data is entered into the system trace buffers using the **trace** shell subcommand. If the shell subcommand was not invoked prior to using the **trace** subcommand, the system trace buffers are empty. KDB kernel debugger trace buffers can be viewed by specifying the **-K** option. Data is entered into the KDB kernel debugger trace buffers using the **trcstart** kdb subcommand. When buffered entries are displayed, they are shown in reverse order (most recent first).

The default trace entry output format is a simplified and short view of the trace record. To view trace entries in a more verbose format, use the **-v** parameter.

The **trace** subcommand is not meant to replace the shell shell subcommand in *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions Volume 2*, which formats the data in more detail. The **trace** subcommand is a facility for viewing system trace data from KDB kernel debugger or kdb command on a dump. The `trcdead` and `trcrpt` shell subcommands are useful in working with trace buffers contained in a system dump.

Aliases
No aliases.
Example
The following example dumps the event buffer on channel 2, related to Thread ID 14539 for an active system trace (the trace is not initiated by KDB kernel debugger):

KDB(0)> trace -c 2 -t 14539

The following is an example of how to initiate a KDB kernel debugger trace and use the `trace` command to view any resulting trace entries:

KDB(0)> trcstart
Kernel Trace initialized successfully
Quit out of kdb, for tracing to continue
KDB(0)> q
Debugger entered via keyboard.
KDB(0)> trcstop
Kernel trace stopped successfully
KDB(0)> trace -K -v
Current entry is #1522 of 1522 at F100009E1460D088
  Hook ID: KERN_SLIH (00000102)  Hook Type: 0
  ThreadIdent: 0000A00B
  Subhook ID/HookData: 0000
  Data Length: 0008 bytes
  D0: 0049BDF0
Current entry is #1521 of 1522 at F100009E1460D068
  Hook ID: KERN_FLIH (00000100)  Hook Type: Timestamped 8000
  ThreadIdent: 0000A00B
  Subhook ID/HookData: 0005
  Data Length: 0008 bytes
  D0: 00028B10
Current entry is #1520 of 1522 at F100009E1460D050
  Hook ID: KERN_SLIH (00000102)  Hook Type: 0
  ThreadIdent: 00008009
  Subhook ID/HookData: 0000
  Data Length: 0008 bytes
  D0: 0049BDF0
**trcstart subcommand**

**Purpose**
The **trcstart** subcommand starts system trace for the KDB kernel debugger. This command cannot be used with the **kdb** command. For more information and to see an example, see "trace subcommand" on page 384.

**Syntax**

```
trcstart [ -f | -l ] [ -j events ] [ -k events ] [ -p ]
```

**Parameters**

- `-f`  Logs only the first trace buffers collected.
- `-l`  Logs only the last trace buffers collected.
- `-j events`  Traces only the specified events. The events must be separated by commas.
- `-k events`  Traces events that are not specified. The events must be separated by commas.
- `-p`  Places the processor identifier in each trace event. This parameter can only be used for 64-bit kernels.

The trace daemon starts a system trace. When the trace is viewed with the **trace** subcommand, the most recently-gathered data is shown. The `-l` parameter is the default.

**Aliases**
No aliases.

**Example**

To trace hooks 101 and 104, use the following subcommand:

```
trcstart -j 101,104
```
trcstop subcommand

Purpose
The trcstop subcommand stops a kdb trace. This command cannot be used with the kdb command. For more information and to see an example, see "trace subcommand" on page 384.

Syntax
trcstop

Parameters
No parameters.

Aliases
No aliases.

Example
See "trace subcommand" on page 384.
mtrace subcommand

Purpose
The mtrace subcommand displays information about the Lightweight Memory Trace (LMT).

Syntax
mtrace [ -C [ CPU_list | all ] [ -d addr [, size ] ] [-t TID ] [ -j Event [, Event ] ] [ -v ] [ rare | common | all ]

Parameters
-C [ CPU_list | all ] Specifies the logical IDs of processors in decimal format. The CPU_list parameter is a comma-separated list of logical processor IDs. The keyword all is used to identify all active processors. If the mtrace command requires specified processors and none is provided, -C all is assumed.
-d addr [, size ] Specifies the memory trace buffer address and size.
-t TID Thread ID filter. A thread is identified by its thread ID in hexadecimal format. When the mtrace command is used to display buffer contents, this parameter is used to display only LMT events related to the specified thread. Otherwise, this parameter is not valid.
-j Event [, Event ] Hook ID filter. A hook is identified by its hook ID in hex format. When the mtrace command is used to display buffer contents, this parameter is used to display only LMT events for the specified hook ID. Otherwise, this parameter is not valid. A maximum of 128 hooks can be specified.
-v Displays events using a verbose output format. This option is only valid when the mtrace command is used to display buffer contents.
rare | common | all Specifies the buffer types to be displayed.

If LMT is in disabled mode, only general LMT information can be displayed. If the kdb command is invoked on a live kernel, trace events in buffers cannot be displayed.

If no options are specified, the mtrace command displays general information about LMT (the contents of the mtrc structure).

If the -C parameter is specified with a single processor and no buffer type, information for the common and rare buffer types on the specified processor is provided.

If the -C parameter is specified with one or more processors and a buffer type, trace events recorded in the specified buffer of the specified processors are displayed, with the most recent events displayed first.

If the -d parameter is specified, trace events recorded in the buffer at the specified address and of the specified size are displayed. Use the -d parameter to display memory trace events saved in the dmp_minimal area of a system dump.

The default trace entry output format is a simplified and short view of the trace record. To view trace entries in a more verbose format, use the -v parameter.

Aliases
mtrc

Example
The following example displays memory trace buffer information for processor 0 using the alias mtrc subcommand:
The following example displays the trace event buffer for processor 0 in a verbose output format using the alias `mtrc` subcommand:

```
KDB(0)> mtrc -C 0
Display content of buffer: mtrcq @ F10008000FF99040
Current entry at @ F100011870064088
   Hook ID: KERN_SLIH (00000102)    Hook Type:
          ThreadId: 00000205
          Subhook ID/HookData: 0000
          Data Length: 0008 bytes
          D0: 0000000003EC2050           .............

Current entry at @ F100011870064068
   Hook ID: KERN_FLIH (00000100)    Hook Type: Timestamped
          ThreadId: 00000205
          Subhook ID/HookData: 0005
          Data Length: 0028 bytes
          D0: 000000000002E36C           .............
          D1: 0000000000000000           .............
          D2: F0000002FF476000           .............
          D3: 0000000000000000           .............
          D4: 0000000000000000           .............
```

The following example merges and displays all entries using the alias `mtrc` subcommand:

```
KDB(0)> mtrc all
```

The following example displays events in both the common and rare buffer types on processor 0 and 3 with thread ID 1893 and hook 0x100, 0x200 and 0x3B7 using the alias `mtrc` subcommand:

```
KDB(0)> mtrc -C 0,3 -t 1893 -j 100,200,3B7 all
```

The following example displays summary information:

```
KDB(0)> mtrc
```

```
MTRC @ 00000000011732B8
mt_magic........... mtrc
mt_state.......... 00000000 ENABLED
mt_flags.......... 00000000
mt_lock ........... 0000000000000000
mt_bufsize[COM]... 0000000000098000
mt_bufsize[RAR]... 0000000000065000
mt_reqbufsize[COM] FFFFFFFF00000000
mt_reqbufsize[RAR] FFFFFFFF00000000
mt_cdtsize....... 000000000007E827B
mt_cdt........... 0000000000000000
mt_wait.......... FFFFFFFF00000000
```
cdt subcommand

Purpose
The cdt subcommand displays data in a system memory dump.

Note: This subcommand is only available within the kdb command. It is not included in the KDB kernel debugger.

Syntax
cdt [-d] [index] [entry]

Parameters
-d Indicates that the dump routines in the /usr/lib/ras/dmprtns directory are used to display data from component dump tables.
index Indicates the component dump table to be viewed. This must be a decimal value.
entry Indicates the data area of the indicated component to be viewed. This must be a decimal value.

Any component dump area can be displayed. With no parameters, all component dump table headers are displayed. If an index is specified, the component dump table header and associated entries are displayed. If both an index and an entry are specified, the data for the indicated area is displayed in both hexadecimal and ASCII. If the -d flag is specified, the dump formatting routines, if any, for the specified component are invoked to format the data in the component data areas.

Aliases
No aliases.

Example
The following is an example of how to use the cdt subcommand:

(0)> cdt
1) CDT head name proc, len 001D00E8, entries 96676
2) CDT head name thrd, len 003ABE4C, entries 192489
3) CDT head name errlg, len 00000054, entries 3
4) CDT head name bos, len 00000040, entries 2
5) CDT head name vmm, len 00000004, entries 30
6) CDT head name sscsidd, len 0000000C, entries 3
7) CDT head name dptSR, len 00000005, entries 3
8) CDT head name scdisk, len 00000130, entries 14
9) CDT head name lvm, len 00000040, entries 2
10) CDT head name SSA$, len 000000A4, entries 7
11) CDT head name SSAES, len 00000054, entries 3
12) CDT head name ssagateway, len 00000007, entries 5
13) CDT head name tty, len 00000008, entries 4
14) CDT head name sio_dd, len 00000005, entries 3
15) CDT head name netstat, len 0000000E, entries 10
16) CDT head name ent2104x, len 00000005, entries 3
17) CDT head name cstkdd, len 00000007, entries 5
18) CDT head name atm_dd_charm, len 00000004, entries 2
19) CDT head name ssa$iK, len 0000002AC, entries 33
20) CDT head name SSADS, len 00000004, entries 2
21) CDT head name osi_frame, len 0000002C, entries 1
(0)> cdt 12
12) CDT head name ssagateway, len 0000007C, entries 5

CDT 1 name HashTbl addr 000000001A25C0, len 00000040
CDT 2 name CfgdAdap addr 000000001A0E044, len 00000004
CDT 3 name OpenAdap addr 000000001A0E048, len 00000004
CDT 4 name LockWord addr 000000001A0E04C, len 00000004
CDT 5 name ssa0 addr 0000000001A20D000, len 00000B88
(0)> cdt -d 12 4
12) CDT head name ssagateway, len 0000007C, entries 5
CDT 4 name LockWord addr 0000000001A0E04C, len 00000004
01A0E04C: FFFFFFFF ....
errpt subcommand

Purpose
The `errpt` subcommand displays system error log entries that were not processed by the error daemon. The entries are displayed in ascending chronological order with the oldest first.

Syntax
```
errpt
```

Parameters
No parameters.

Aliases
No aliases.

Example
The following is an example of how to use the `errpt` subcommand:
```
KDB(6)> errpt
ERRORS NOT READ BY ERRDEMON (ORDERED CHRONOLOGICALLY):

Error Record:
erec_flags .............. 0
erec_len .............. 40
erec_timestamp .......... 4034EA04
erec_rec_len .......... 20
erec_dupcount .......... 0
erec_duptime1 .......... 0
erec_duptime2 .......... 0
erec_rec.error_id ...... 2BFA76F6
erec_rec.resource_name .. SYSPROC
 00000000 00000000 00000000 ............

Error Record:
erec_flags .............. 0
erec_len .............. 834
erec_timestamp .......... 4036203C
erec_rec_len .......... 814
erec_dupcount .......... 0
erec_duptime1 .......... 0
erec_duptime2 .......... 0
erec_rec.error_id ...... BFE4C025
erec_rec.resource_name .. Sysplanar0
 2040220 00000000 00000000 00000000 ............
 000020FF 00000001 00000000 00000000 ............
49424000 55312E31 302D0531 20433200 IBM.U1.10-P1-C2.
 00020000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
 00000000 00000000 00000000 00000000 ............
```

<snip>
check subcommand

Purpose
The check subcommand runs consistency checkers on kernel data structures.

Syntax
check

check ? | -?

check [-h] CheckerName

check [-v] [-l level] [-n count] CheckerName [ .SuffixName ] [ EffectiveAddress ]

check [-e] [-v] [-l level] CheckerName [ .SuffixName ] EffectiveAddress

Parameters

CheckerName Specifies the name of the checker to run. Run the check command with no parameters to display the list of known checkers.

SuffixName Specifies which of the suffixes of the given checker to run. Run the check command with the -h parameter to display the list of known suffixes for a given checker.

Effective Address Specifies the effective address of the element to be validated or the effective address of the first element to be validated for lists. Symbols, hexadecimal values, or hexadecimal expressions can be used to specify the effective address.

-e Specifies that only one element should be checked. This is equivalent to -n 1. An effective address must be specified if the -e parameter is specified.

-h Displays help for each suffix of the specified checker.

-l level Specifies the checking level the checker should use. This is a decimal value between 0 and 9. A value of 9 specifies the most detailed checking level and a value of 0 specifies no checking. The default value is 3 (light level) unless the -e flag is specified, in which case the default value is 7 (detailed level).

-n count Specifies the number of elements (count is a decimal value) to validate.

-v Specifies that the checker should run in verbose mode and display additional information if the checker supports this option.

Aliases
No aliases.

Example
1. To display the list of known checkers, type the following:
   check
   Output similar to the following displays:
   Please specify a checker name:

   Kernel Checkers Description
   ----------------- -----------------------------------------------
   proc Validate proc and pvproc structures
   thread Validate thread and pvthread structures

   Kernext Checkers Description
   ---------------------------

2. To display detailed help for a specified checker, type the following:
   check -h proc
Output similar to the following displays:

Checker 'proc' is used to validate pvproc and proc structures:

- `proc` check the global pvproc process table
- `proc <addr>` check a single pvproc
- `proc.pv_db <addr>` check a list of pvproc linked by pv_dbnext
- `proc.pv_sched <addr>` check a list of pvproc linked by pv_sched_next/back
- `proc.pv_siblings <addr>` check a list of pvproc linked by pv_siblings
- `proc.pv_pgrp <addr>` check a list of pvproc linked by pv_pgrpl/pv_pgrpb
- `proc.pv_ttyl <addr>` check a list of pvproc linked by pv_ttyl
- `proc.pv_crid <addr>` check a list of pvproc linked by pv_cridnext

For each element, both pvproc and associated proc structure are validated
- `<addr>` should be the address of a pvproc structure (not a proc structure)

3. To run proc checker to validate the entire process table, type the following:

   check -l 7 proc

   Output similar to the following displays. In this example, a corruption is found in a flag.

   Corruption found in pvproc.pv_flag: F100020E0000A400+0100 | RASCHK_BAD_BITMASK | Invalid flags

4. To run proc checker to perform a detailed check on a single process, type the following:

   check -e -l 7 proc pvproc+006800

5. To run proc checker to validate the first five elements of a list of processes linked by the pv_siblings field starting at pvproc+00AC00 in verbose mode, type the following:

   check -l 7 -n 5 -v proc.pv_siblings pvproc+00AC00

   Output similar to the following displays:

   Last element checked: F100020E0000AC00 <pvproc+00AC00>
   Last element checked: F100020E0000C000 <pvproc+00C000>
   Last element checked: F100020E0000BC00 <pvproc+00A400>
   Corruption found in pvproc.pv_flag: F100020E0000A400+0100 | RASCHK_BAD_BITMASK | Invalid flags
   Last element checked: F100020E0000BC00 <pvproc+00BC00>
   Last element checked: F100020E0000B000 <pvproc+00B0000>
ctctrl subcommand

Purpose
The ctctrl subcommand displays information related to Component Trace (CT).

Syntax
ctctrl [[ -r ] | [ -D ] | [ -j Event [, Event ] ] ] [ -l alias name ] [ -c component name ] [ -t tid ] [ -v ]

Parameters
- **-c component name** Specify a component by name.
- **-l alias name** Specify a component by alias.
- **-r** Applies the query recursively to all children components.
- **-D** Displays component trace buffer events of a specified component.
- **-j Event [, Event]** Hook ID filter. A hook is identified by its hook ID in hexadecimal format. When the ctctrl command is used to display buffer contents, this parameter displays only events for the specified hook IDs. Otherwise, this parameter is not valid. A maximum of 128 hooks can be specified.
- **-t tid** Thread ID filter. A thread is identified by its thread ID in hexadecimal format. When the ctctrl command is used to display buffer contents, this parameter is used to display only events related to the specified thread. Otherwise, this parameter is not valid.
- **-v** Displays events using a verbose output format. This option is only valid when displaying trace events of a specified component.

By default, without any parameters, ctctrl displays the settings of the base components. Specific components can be specified with the -l parameter or the -c parameter. To display the trace entries, use the -D parameter. The format used is the same as that of the trace command.

The default trace entry output format is a simplified and short view of the trace record. To view trace entries in a more verbose format, use the -v parameter.

The display of trace entries are not supported in the kdb command on a live system because of synchronization problems (buffers continue to be filled when being read, buffer resize operation can occur, and so on). The kdb command can display trace entries when working on a dump. Moreover, in user space, the memory buffers can be read and examined with the ctctrl command or the trcrpt command.

Aliases
No aliases.

Example
The following example displays the events in the net.route component with hook ID 0x617.
KDB(0)> ctctrl -D -c net.route -j 617

The following is the output of a ctctrl command with no arguments. The parameter information of all of the active base components is displayed:
(0)> ctctrl

Component Name: ethernet
Alias: No
Displaying rasp_trace_block at: F100010030247780
rtb_evec............. 7472636252415361: (EYEC_RAST)
rtb_flags............. EO: (Memory Trace: ON System Trace: ON)
rtb_memlevel_internal.. 1
rtb_syslevel_internal.. 7
The following example displays the parameter information about the socket component and all of its subcomponents:

(0)> ctctrl -c socket -r

Component Name: socket
Alias: No
Displaying rasp_trace_block at: F10001002D432C80
rtb_eyec.......................... 7472636252415361: (EYEC_RAST)
rtb_flags.......................... E0: (Memory Trace: ON System Trace: ON)
rtb_memlevel_internal............. 1
rtb_syslevel_internal............. 7
rtb_bufbase............. F10001002D646000
rtb_bufend............. F10001002D64FFC0
rtb_bufinptr............. F10001002D6517E0
rtb_busize............. 40960
rtb_bufwrap_t0............. 0000A41C5E0A85CE
rtb_bufwrap_t1............. 0000AAEDA153D588

Component Name: socket.so_unix
Alias: No
Displaying rasp_trace_block at: F10001002D432E80
rtb_eyec.......................... 7472636252415361: (EYEC_RAST)
rtb_flags.......................... E0: (Memory Trace: ON System Trace: ON)
rtb_memlevel_internal............. 1
rtb_syslevel_internal............. 7
rtb_bufbase............. F10001002D650000
rtb_bufend............. F10001002D6527C0
rtb_bufinptr............. F10001002D6517E0
rtb_busize............. 10240
rtb_bufwrap_t0............. 0000000000000000
rtb_bufwrap_t1............. 000051DDCBBE793F

396  KDB Kernel debugger and kdb command
Chapter 35. Lock subcommands

The subcommands in this category can be used to display information about locks and to check the system for deadlocks. These subcommands include the following:

- `lk`
- `slk`
- `clk`
- `dlk`
- `dla`
lk, slk, clk, and dlk subcommands

Purpose
The lk (display lock_t lock), slk (display simple lock), clk (display complex lock) and dlk (display dist lock) subcommands can be used to display information about locks.

Note: The dlk subcommand is only available with the 64-bit kernel.

Syntax
lk [ lock_address ]
slk [-q] [ lock_address ]
clk [-q] [ lock_address ]
dlk [-q] [ lock_address ]

Parameters
lock_address Specifies the address of the lock. Symbols, hexadecimal values, and hexadecimal expressions can be used to specify the address.
-q Keeps instrumentation information from displaying. If instrumentation is set at boot time and the -q option is not entered, slk, clk, and dlk show instrumentation information.

If no parameter is given, a default list of locks is displayed.

Aliases
No aliases.

Example
Instrumentation is set to on by using the -L option of the bosboot command. The following is an example of how to use the lk, slk, clk and dlk subcommands with instrumentation set to on:

KDB(0)> lk
//show status of default list of locks
Major Locks:
acct_lock Available 03E6B180
lock F100109E0866D280 INTERLOCK cpu_owner............. 00000000 θ F100109E0866D280
audit_lock Available
audit_q_lock Available
audit_w_lock Available
03BC50F8 Available
bio_lock Available
bus_reg_lock Available
cio_lock Available
clist_lock Available
ccons_lock Available
core_lock Available
cred_alloc_lock Available
cs_lock Available
ctrace_lock Available
devswlock
lock F100109E0802AF30 thread_owner............. 0802AF30 θ pvthread+7802A00
dil_lock
Available
(0)> more (^C to quit) ? ^C
398 KDB Kernel debugger and kdb command
Chapter 35. Lock subcommands

KDB(0)> 1k acct_lock //show lock_t lock acct_lock
acct_lock Available
KDB(0)> nm acct_lock //show address of acct_lock
Symbol Address: 0149CF00
   TOC Address: 0149A200
KDB(0)> 1k 0149CF00 //show acct_lock using address
acct_lock Available
KDB(0)> s lk cio_lock //show simple lock cio_lock
cio_lock Available
Instrumented lock...... @ F100109E0801A0E0

..............lockname: FFFFFFFF
KDB(0)> slk -q cio_lock //show cio_lock without instrumentation
cio_lock Available
KDB(0)> cl k jfs_quota_lock //show complex lock jfs_quota_lock
jfs_quota_lock Available
Instrumented lock...... @ F100109E0C006EA0

..............lockname: FFFFFFFF
KDB(0)> cl k -q jfs_quota_lock //show jfs_quota_lock without instrumentation
jfs_quota_lock Available
KDB(0)> dl k wlm_classes_lock //show dist lock wlm_classes_lock
wlm_classes_lock
mutex............. F100109E0C000050 write owner ........ 0000000000000000
writer await........... FFFFFFFFFFFFFFFF count............. 0000000000000000
writer wait reader..... FFFFFFFFFFFFFFFF count............. 0000000000000000
reader await........... FFFFFFFFFFFFFFFF count............. 0000000000000000
readers active........ 0000000000000000 reader counter..... 0 F10010F004056080
node interlace........ 0000000000000200 instrumented..... 0 F100109E0B017ED0

cpg shift.................. 00 cpu groups......................... 02
group shift............ 01 grp mask......................... 01

Group counters:
SRAD ID: 0000
Group 0............. 0000000000000000 @ F10010F004056080
Group 01............ 0000000000000000 @ F10010F004056100
Instrumented lock...... @ F100109E0B017ED0

..............lockname: 00000000
KDB(0)> dl k -q wlm_classes_lock //show wlm_classes_lock without instrumentation
wlm_classes_lock
mutex............. F100109E0C000050 write owner ........ 0000000000000000
writer await........... FFFFFFFFFFFFFFFF count............. 0000000000000000
writer wait reader..... FFFFFFFFFFFFFFFF count............. 0000000000000000
reader await........... FFFFFFFFFFFFFFFF count............. 0000000000000000
readers active........ 0000000000000000 reader counter..... 0 F10010F004056080
node interlace........ 0000000000000200 instrumented..... 0 F100109E0B017ED0

cpg shift.................. 00 cpu groups......................... 02
group shift............ 01 grp mask......................... 01

Group counters:
SRAD ID: 0000
Group 0............. 0000000000000000 @ F10010F004056080
Group 01............ 0000000000000000 @ F10010F004056100
**dla subcommand**

**Purpose**
The `dla` subcommand checks the system for deadlocks and displays details about threads waiting for locks.

**Note:** The `dla` subcommand is only available with the `kdb` command.

**Syntax**
```
dla [ {-p [cpu]} | tid ]
```

**Parameters**
- `-p` Reports only on the locks waited on by the specified processor. If no processor is specified, reports on all of the processors.
  - `cpu` Specifies the cpu number.
  - `tid` Report on locks waited on by the thread specified by this thread identifier.

If no arguments are given, the `dla` subcommand analyzes the system for deadlocks. The `dla` subcommand also shows details on any thread waiting for a lock.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `dla` subcommand:

```
(0)> dla
No deadlock, but chain from tid 42C5, that waits for the first line lock, owned by Owner-Id that waits for the next line lock, and so on ...

<table>
<thead>
<tr>
<th>LOCK NAME</th>
<th>ADDRESS</th>
<th>OWNER-ID</th>
<th>LOCK STATUS</th>
<th>WAITING FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptrace_lock</td>
<td>0x00000000006E9898</td>
<td>Tid 1B37</td>
<td>0x20000000</td>
<td>slock_ppc</td>
</tr>
</tbody>
</table>

No deadlock, but chain from tid 53AF, that waits for the first line lock, owned by Owner-Id that waits for the next line lock, and so on ...

<table>
<thead>
<tr>
<th>LOCK NAME</th>
<th>ADDRESS</th>
<th>OWNER-ID</th>
<th>LOCK STATUS</th>
<th>WAITING FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptrace_lock</td>
<td>0x00000000006E9898</td>
<td>Tid 1B37</td>
<td>0x20000000</td>
<td>slock_ppc</td>
</tr>
</tbody>
</table>

No deadlock found
```

```
(0)> dla 42C5
No deadlock, but chain from tid 42C5, that waits for the first line lock, owned by Owner-Id that waits for the next line lock, and so on ...

<table>
<thead>
<tr>
<th>LOCK NAME</th>
<th>ADDRESS</th>
<th>OWNER-ID</th>
<th>LOCK STATUS</th>
<th>WAITING FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptrace_lock</td>
<td>0x00000000006E9898</td>
<td>Tid 1B37</td>
<td>0x20000000</td>
<td>slock_ppc</td>
</tr>
</tbody>
</table>

No deadlock found
```

```
(0)> dla -p 0
No locks being waited on for processor 0
```

```
(0)> dla -p
No deadlock found
```
Chapter 36. Network subcommands

The subcommands in this category are used to print network information. These subcommands include the following:

- ifnet
- tcb
- udb
- sock
- tcpcb
- mbuf
- netm
- sockinfo
- ndc
- nsdbg
- netstat
- route
- rtentry
- rxnode
- tcpdbg

© Copyright IBM Corp. 1997, 2006
ifnet subcommand

Purpose
The ifnet subcommand prints network interface information.

Syntax
ifnet [slot | effectiveaddress]

Parameters
- slot: Specifies the slot number within the ifnet table for which data is to be displayed. This value must be a decimal number.
- effectiveaddress: Specifies the effective address of an ifnet entry to display.

If no parameter is specified, information is displayed for each entry in the ifnet table. Display data for individual entries by specifying either a slot number or by specifying the address of the entry.

Aliases
No aliases.

Example
The following is an example of how to use the ifnet subcommand:

KDB(0)> ifnet 1
SLOT 1 ---- IFNET INFO ----(@ F10006000CDF2000)----
  name........ en0    unit....... 00000000    mtu........ 00000DC    flags....... 5E080863
    (UP|BROADCAST|NOTRAILERS|RUNNING|SIMPLEX|NOECHO|BPF|GROUP_ROUTING...
    ...|64BIT|CHECKSUM_OFFLOAD|PSEG|CANTCHANGE|MULTICAST)
    timer....... 00000000    metric...... 00000000
    address: 9.53.85.113    dest address: 9.53.85.255
    netmask: 255.255.255.0    bk-ptr: F10006000CDF2000
    rentry: 0    ifa_flags: 1
    ifa_refcnt: 5    ifa_rrequest: 0

  init()...... 00000000  output().... 03DE2160  start()..... 00000000
  done()...... 00000000  ioctl1().... 03DE2178  reset()..... 00000000
  watchdog(). 00000000  ipackets.... 00000376  ierrors..... 00000000
  opackets.... 00000247  oerrors..... 00000000  collisions.. 00000000
  next.......0000000000000000  addrlen............ 00000006
  type....... 00000006 (ETHER)
  hdrlen..... 0000000E  index....... 00000002
  lastchange.. 40B36BE3  sec 00030003  usec

  ibytes...... 00048FDC  obytes...... 0001BD0C  imcasts...... 00000000
  omcasts..... 00000007  iqdrops.... 00000000  noproto..... 00000000
  baudrate.... 00A00000  arpdrops.... 0000000000000000
  ifbufminsize 00000000  devno....... 00000000  chan....... 00000000
  multiaddrs..0F100061000BF0F8  tapctl....0000000000000000
  ndtype...... 02032800

KDB Kernel debugger and kdb command
vipaxface..0000000000000000

KDB(0)>
tcb subcommand

Purpose
The tcb subcommand displays the inpcb structure for TCP connections.

Syntax
tcb [-s | -b index | effectiveaddress]

Parameters
- **-s** Specifies the bucket number within the tcb hash table. All tcb entries in this bucket are displayed in detail. The -b indicates that the number that follows is a bucket number and not an effective address.
- **-b index** Specifies the effective address of a tcb entry to display in detail.
- **effectiveaddress** Specifies the effective address of a tcb entry to display in detail.

If no parameters are specified, detailed information is displayed for all entries in the tcb table. A summary of all entries or detailed information for a specific entry can be displayed with the appropriate parameters.

Aliases
No aliases.

Example
The following is an example of how to use the tcb subcommand:

KDB(0)> tcb -s
SLOT 13 TCB -------- INPCB INFO ----(@ F100061000BF5A58)----
SLOT 21 TCB -------- INPCB INFO ----(@ F100061000BF7258)----
SLOT 23 TCB -------- INPCB INFO ----(@ F100061000BF7A58)----
SLOT 25 TCB -------- INPCB INFO ----(@ F1000610004C0A58)----
SLOT 37 TCB -------- INPCB INFO ----(@ F100061000BF2258)----
SLOT 111 TCB ------- INPCB INFO ----(@ F10006100039BA58)----
SLOT 512 TCB ------- INPCB INFO ----(@ F100061000BF5258)----
SLOT 513 TCB ------- INPCB INFO ----(@ F100061000BF6A58)----
SLOT 514 TCB ------- INPCB INFO ----(@ F100061000BF6258)----
SLOT 6864 TCB------- INPCB INFO ----(@ F100061000DB4258)----
SLOT 8269 TCB------- INPCB INFO ----(@ F1000610003F6A58)----
SLOT 8288 TCB------- INPCB INFO ----(@ F1000610003F6A58)----
SLOT 8289 TCB------- INPCB INFO ----(@ F100061000C1AA58)----
SLOT 9090 TCB------- INPCB INFO ----(@ F100061000BF2A58)----
KDB(0)> tcb F100061000BF2258 //tcb address in slot 37
SLOT 37 TCB ------- INPCB INFO ----(@ F100061000BF2258)----
   next........0000000000000000 prev........0000000000000000
   head........000000000000003E63780 faddr_6.....0F100061000BF2278
   iflowinfo...00000000 fport........00000000 fatype......00000000
   oflowinfo...00000000 lport........0000025 latype......00000000
   laddr_6.....0F100061000BF2290 socket.......0F100061000BF2000
   ppcb.........0F100061000BF2360 route_6......0F100061000BF2280
   ifa........0000000000000000 flags.......00000400
   proto.......00000000 tos........00000000 ttl........000003C
   rcvttl......00000000 rcvif.......0000000000000000
   options.....0000000000000000 refcnt......00000000
   lock........0000000000000000 rc_lock.....0000000000000000
   moptions....0000000000000000 hash.next...0F10006000C606378
   hash.prev...0F10006000C606378 timewait.nxt0000000000000000
   timewait.prv0000000000000000 inp_v6opts 0000000000000000
   inp_pmtu....0000000000000000
   --- SOCKET INFO ----(@ F100061000BF2000)----
type........ 0001 (STREAM)
opts........ 0006 (ACCEPTCONN|REUSEADDR)
linger...... 0000 state....... 0080 (PRIV)
pcb.....@F100061000BF2258 proto.....0000000000000000E5A7A8
lock....@F1000610002D7640 head.....000000000000000000000000
q0......000000000000000000000000 q.......000000000000000000000000
q0len...... 0000 qlen........ 0000 qlimit...... 03E8
timeo....... 0000 error....... 0000 special..... 0A0B
pgid....@0000000000000000 oobmark. 0000000000000000

snd:cc...... 000000000000000000000000 hiwat... 000000000000000000000000
mbcnt... 000000000000000000000000 mbmax... 000000000000000000000000
lowat... 000000000000000000000000 mb... 000000000000000000000000
sel.....000000000000000000000000 events...... 0000
iodone.. 00000000 ioargs... 000000000000000000000000
lastpkt.000000000000000000000000 wakeone. FFFFFFFFFFFFFF
timer....000000000000000000000000 timeo... 00000000
flags....... 0000 ()
wakeup.. 00000000 wakearg.000000000000000000000000

lockwtg. FFFFFFFFFFFFFF

MBUF LIST

rcv:cc...... 000000000000000000000000 hiwat... 000000000000000000000000
mbcnt... 000000000000000000000000 mbmax... 000000000000000000000000
lowat... 000000000000000000000000 mb... 000000000000000000000000
sel.....000000000000000000000000 events...... 0001
iodone.. 00000000 ioargs... 000000000000000000000000
lastpkt.000000000000000000000000 wakeone. FFFFFFFFFFFFFF
timer....000000000000000000000000 timeo... 00000000
flags....... 0008 (SEL|NOTIFY)
wakeup.. 00000000 wakearg.000000000000000000000000

lockwtg. FFFFFFFFFFFFFF

MBUF LIST

tpcb....000000000000000000000000 fdev_ch.@F10006000C3E16C0
sec_info000000000000000000000000 qos.....000000000000000000000000
gidlist.000000000000000000000000 private.000000000000000000000000
uid..... 00000000 bufsize. 00000000 threadcnt00000000
nextfree@000000000000000000000000 siguid.. 00000000 siguid. 000000000000000000000000
sndtime. 000000000000000000000000 sec 000000000000000000000000 usec
rcvtime. 000000000000000000000000 sec 000000000000000000000000 usec
saioq... 000000000000000000000000 saioqd.000000000000000000000000
accept.. FFFFFFFFFFFFFF frctime 00000000
isnoflgs 00000000 ()
rclen.. 000000000000000000000000 frclenback000000000000000000000000
frcassoc000000000000000000000000 frcbck 000000000000000000000000
iodone... 00000000 iodonefl 00000000 ()
ioarg... 000000000000000000000000 refcnt.. 000000000000000000000000

proc/fd: 29/19
KDB(0)>
udb subcommand

Purpose
The udb subcommand displays the inpcb structure for UDP connections.

Syntax
udb [-s | -b index | effectiveaddress]

Parameters
-s Displays a one line summary of every udb entry.
-b index Specifies the bucket number within the udb hash table. All udb entries in this bucket are
displayed in detail. The -b indicates that the number that follows is a bucket number and not an
effective address.
effectiveaddress Specifies the effective address of a udb entry to display in detail.

If no parameters are specified, detailed information is displayed for all entries in the udb table. Display a
summary of all entries or detailed information for a specific entry by specifying the appropriate parameters.

Aliases
No aliases.

Example
The following is an example of how to use the udb subcommand:

KDB(0)> udb -s
SLOT 13 UDB --------- INPCB INFO ----(0 F100061000BF3000)----
SLOT 37 UDB --------- INPCB INFO ----(0 F100061000BF3200)----
SLOT 111 UDB --------- INPCB INFO ----(0 F100061000BF6600)----
SLOT 123 UDB --------- INPCB INFO ----(0 F100061000BF6800)----
SLOT 123 UDB --------- INPCB INFO ----(0 F100061000BF6C00)----
SLOT 135 UDB --------- INPCB INFO ----(0 F100061000BF6800)----
SLOT 514 UDB --------- INPCB INFO ----(0 F100061000BF6800)----
SLOT 518 UDB --------- INPCB INFO ----(0 F100061000BF6C00)----

KDB(0)> udb F100061000BF6B00 //udb address in slot 111

KDB Kernel debugger and kdb command
Chapter 36. Network subcommands
sock subcommand

Purpose
The sock subcommand prints socket structure for UDP and TCP sockets

Syntax
sock [-d] [tcp | udp] [effectiveaddress]
sock -s [tcp | udp]
sock -f

Parameters
-d Suppresses the display of send and receive buffer information for a socket.
-s Displays a one-line summary of every socket. If the optional tcp or udp parameter is used with
-s, displays a summary of only the specified socket types.
-f Displays the "free page list".
tcp Displays socket information for TCP blocks only.
udp Displays socket information for UDP blocks only.
effectiveaddress Specifies the effective address of a particular socket structure to display.

If no parameter is specified, detailed information is displayed for every allocated TCP or UDP socket on
the system. The displayed information can be restricted to only a particular socket type by using the tcp
parameter or the udp parameter. Specifying the effective address of a particular socket structure, limits
the display to that structure.

Aliases
No aliases.

Example
The following is an example of how to use the sock subcommand:

408 KDB Kernel debugger and kdb command
pcb...@F1000610003F0258  proto...@00000000003E427A8
lock...@00000000003F0258  head...@0000000000000000
q0......@0000000000000000  q......@0000000000000000
q0len...... 0000  qlen...... 0000  qlimit...... 03E8
timeo...... 0000  error...... 0000  special...... 0A08
pgid...@0000000000000000  oobmark. 0000000000000000

snd:cc...... 0000000000000000  hiwat... 00000000000000000E000
mbcnt... 0000000000000000  mbmax... 0000000000038000
lowat... 0000000000001000  mb...... 0000000000000000
sel...... 0000000000000000  events...... 0000
iodone. 00000000  ioargs...@0000000000000000
lastpkt.0000000000000000  wakeone. FFFFFFFFFFFFFF
timer... 0000000000000000  timeo... 00000000
flags....... 0000 ()
wakeup... 0000000  wakearg.@0000000000000000
lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

rcv:cc...... 0000000000000000  hiwat... 00000000000000000E000
mbcnt... 0000000000000000  mbmax... 0000000000038000
lowat... 0000000000001000  mb...... 0000000000000000
sel...... 0000000000000000  events...... 0001
iodone. 00000000  ioargs...@0000000000000000
lastpkt.0000000000000000  wakeone. FFFFFFFFFFFFFF
timer... 0000000000000000  timeo... 00000000
flags....... 0008 (SEL|NOTIFY)
wakeup... 0000000  wakearg.@0000000000000000
lockwtg. FFFFFFFFFFFFFFFF

MBUF LIST

tpcb....@0000000000000000  fdev_ch.@F10006000CE0F600
sec_info@0000000000000000  qos......@0000000000000000
gidlist.@0000000000000000  private.@0000000000000000
uid.... 00000000  bufsize. 00000000  threadcnt00000000
nextfree@0000000000000000  siguid.. 00000000  sigprv. 00000000
sndtime. 0000000000000000  sec 0000000000000000  usec
rcvt ime. 0000000000000000  sec 0000000000000000  usec
saioq...0000000000000000  saiqhd.0000000000000000
accept. FFFFFFFFFFFFFFF  frcatime 00000000
isnoflgs 00000000 ()
rvcvlen. 0000000000000000  frcaback0000000000000000
frcassoc0000000000000000  frcabckt 0000000000000000
iodone. 00000000  iodonefl 00000000 ()
ioarg...@0000000000000000  refcnt.. 0000000000000000

proc/fd: 98/19
KDB(0)>

Chapter 36. Network subcommands  409
**tcpcb subcommand**

**Purpose**
The `tcpcb` subcommand displays the `tcpcb` structure.

**Syntax**
`tcpcb [-s | effectiveaddress]`

**Parameters**
- `-s` Displays a one-line summary of every tcb entry.
- `effectiveaddress` Specifies the effective address of a `tcpcb` structure to be displayed. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

If no parameter is specified, detailed information is displayed for all `tcpcb` structures. A single `tcpcb` structure is displayed by specifying the effective address of the structure, and a summary of all `tcpcb` structures is displayed by using the `-s` option.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `tcpcb` subcommand:

```
KDB(0)> tcpcb -s
---- TCP -----(inpcb: @ F1000610003F5258)----
---- TCPCB -----(@ F1000610003F5360)----
---- TCP -----(inpcb: @ F1000610003F2A58)----
---- TCPCB -----(@ F1000610003F2B60)----
---- TCP -----(inpcb: @ F1000610003F3258)----
---- TCPCB -----(@ F1000610003F3360)----
---- TCP -----(inpcb: @ F1000610003F3A58)----
---- TCPCB -----(@ F1000610003F3B60)----
---- TCP -----(inpcb: @ F1000610003F4A58)----
---- TCPCB -----(@ F1000610003F4B60)----
---- TCP -----(inpcb: @ F1000610003F4258)----
---- TCPCB -----(@ F1000610003F4360)----
---- TCP -----(inpcb: @ F1000610003F3A58)----
---- TCPCB -----(@ F1000610003F3B60)----
---- TCP -----(inpcb: @ F1000610003F2A58)----
---- TCPCB -----(@ F1000610003F2B60)----
---- TCP -----(inpcb: @ F1000610003F3258)----
---- TCPCB -----(@ F1000610003F3360)----
---- TCP -----(inpcb: @ F1000610003F5258)----
---- TCPCB -----(@ F1000610003F5360)----
---- TCP -----(inpcb: @ F1000610003F5A58)----
---- TCPCB -----(@ F1000610003F5B60)----
---- TCP -----(inpcb: @ F1000610003F5258)----
---- TCPCB -----(@ F1000610003F5360)----
---- TCP -----(inpcb: @ F1000610003F5A58)----
---- TCPCB -----(@ F1000610003F5B60)----
---- TCP -----(inpcb: @ F1000610003F4A58)----
---- TCPCB -----(@ F1000610003F4B60)----
---- TCP -----(inpcb: @ F1000610003F4258)----
---- TCPCB -----(@ F1000610003F4360)----
KDB(0)> tcpcb F1000610003F5360 //address of the first tcpcb structure from above
---- TCP -----(inpcb: @ F1000610003F5258)----
---- TCPCB -----(@ F1000610003F5360)----
    seg_next......@F1000610003F5360  seg_prev......@F1000610003F5360
```
t_softerror... 00000000  t_state...... 00000001 (LISTEN)
t_timer....... 00000000 (TCPT_REXMT)
t_timer....... 00000000 (TCPT_PERSIST)
t_timer....... 00000000 (TCPT_KEEP)
t_timer....... 00000000 (TCPT_2MSL)
t_rxtshift.... 00000000  t_rxtcur...... 00000006  t_dupacks.... 00000000
t_maxseg...... 00000200  t_force....... 00000000
t_flags....... 00000020 (RFC1323|COPYFLAGS)
t_oobflags..... 00000000 ()
t_template.... 0000000000000000  t_inpcb....... 0F1000610003F5258
t_jobc....... 00000000  t_timestamp... 014C0801  snd_una....... 00000000
snd_nxt....... 00000000  snd_up........ 00000000  snd_w11....... 00000000
snd_w12....... 00000000  iss............ 00000000  snd wnd...... 0000000000000000  rcv wnd...... 0000000000000000
rcv nxt....... 00000000  rcv_up........ 00000000  irs............ 00000000
snd wnd_scale. 00000000  rcv wnd scale. 00000000  req_scale_sent 00000000
req_scale/rcvd 00000000  last_ack_sent. 00000000  timestamp_rec. 00000000
timestamp_age. 00000000  rcv_adv....... 00000000  snd_max....... 00000000
snd cwnd....... 000000003FFC000  snd ssthresh.. 000000003FFC000
max_sndwnd.... 0000000000000000  max_sndwnd.... 0000000000000000
max rcvd...... 0000000000000000  max_swnd.... 0000000000000000
max_srtt....... 0000000000000000  max_ssttl....... 0000000000000000
max_rtt....... 0000000000000000  max_rttm.. 0000000000000000
max_rttm........ 0000000000000000  max_rttm........ 0000000000000000
max_ssthresh.. 0000000000000000  max_ssthresh.. 0000000000000000
max_sndwnd.... 0000000000000000  max_sndwnd.... 0000000000000000
max_sndwnd.... 0000000000000000  max_sndwnd.... 0000000000000000
snd in_pipe... 00000000  snd_in_pipe... 00000000
snd_recover... 00000000  snd_recover... 00000000
snd_rtt........ 00000000  snd_rtt........ 00000000
snd_maxseg.... 000000200  snd_in_pipe... 00000000
snd_high...... 00000000  snd_ecn_max... 00000000  snd_ecn_clear. 00000000
t splice with.0000000000000000  t splice_flags 00000000
KDB(0)>
mbuf subcommand

Purpose
The mbuf subcommand displays mbuf information.

Syntax
mbuf [-p I [-a] [-n] [-d]] [effectiveaddress]

Parameters
- **-p** Displays the private mbuf structure pool information.
- **-a** Follows the packet chain. The effectiveaddress parameter is required for this flag.
- **-n** Follows the mbuf structure chain within a packet. The effectiveaddress parameter is required for this flag.
- **-d** Suppresses printing of the mbuf structure data and displays only the mbuf structure header. This is helpful when only the mbuf structure header information is required. The effectiveaddress parameter is required for this flag.

effectiveaddress Specifies the effective address of an mbuf structure to be displayed. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

Display the packet chain and mbuf structure chains within packets by using the -a parameter and the -n parameter.

Aliases
No aliases.

Example
The following is an example of how to use the mbuf subcommand:

KDB(1)> mbuf -p total cluster pools...........00000001 cluster pool 0.........700F8D40
  p_next.................00000000 p_size..........................0000000A p_inuse..................00000001
  m_outcnt..................00000001 m_maxoutcnt...........00000002 next..................70168F00
  tail..................70110F00 m_next..................00000000 m_nextpkt...........71210F00
  m_data..................71164800 m_len..................00000010 m_type............0001
  m_flags.............0041 (M_EXT|M_EXT2) ext_buf...........71164800
  ext_free.............0026C058 ext_size..........................00000400 ext_arg................700F8D40
  ext_forw................70168F2C ext_back..................70168F2C ext_hasxm............00000000
  ext_xmemd...............70168F38 ext_debug............70EF6750
------------------------------------------------------------------
71164800: 7116 4400 3172 D5B8 0000 0000 0000 q.D.Ir..........
netm subcommand

Purpose
The netm subcommand displays the net_malloc event records that are stored in the kernel.

Syntax
netm [-c display_count] [-t type [,.type[,...]]] [-s size [,size[,...]]]

netm -a [effectiveaddress]

netm -i starting_index

netm -e [outstand_mem]

Parameters
- **-c display_count**  Specifies how many of the last records of net_malloc events you want to display.
- **-a**  Displays all records of the net_malloc events.
- **-a effectiveaddress**  Displays only the net_malloc events associated with the specified address.
- **-i starting_index**  Displays the net_malloc events started from the events record numbered starting_index
- **-e**  Displays a list of net_malloc memory addresses that are not freed.
- **-e outstand_mem**  Displays net_malloc events related to the outstanding memory specified by outstand_mem.
- **-t type**  Limits the display to specified types of blocks. Valid values are a subset of those defined in INITKMEMNAMES in the net_malloc.h file.
- **-s size**  Limits the display to specified sizes of blocks.

The netm subcommand is only available after the net_malloc_police attribute is turned on, and the display begins with the latest event. The netm subroutine displays up to 16 stack traces in the net_malloc event.

Aliases
No aliases.

Example
No example.
sockinfo subcommand

Purpose
The sockinfo subcommand displays several different socket-related structures.

Syntax
sockinfo effectiveaddress TypeOfAddress[-d]

Parameters

 effectiveaddress Specifies the effective address of the structure to be displayed.
 TypeOfAddress Identifies the type of structure to which the effective address points. Valid address types are socket, inpcb, rawcb, unpcb, ripcb and tcpcb.
 -d Suppresses the display of send and receive buffer information for a socket.

Aliases

si

Example
The following is an example of how to use the sockinfo subcommand:

KDB(0)> sock tcp -s
--- TCP (inpcb: @F100610003F0258) --- SOCKET @ F100610003F0000
--- TCP (inpcb: @F100610003F1A58) --- SOCKET @ F100610003F1800
--- TCP (inpcb: @F100610003F2258) --- SOCKET @ F100610003F2000
--- TCP (inpcb: @F100610002A6A58) --- SOCKET @ F10061002A60800
--- TCP (inpcb: @F100610003F0A58) --- SOCKET @ F100610003F0800
--- TCP (inpcb: @F10061000435A58) --- SOCKET @ F10061000435800
--- TCP (inpcb: @F100610003FBA58) --- SOCKET @ F100610003FBB800
--- TCP (inpcb: @F100610003F2A58) --- SOCKET @ F100610003F2B800
--- TCP (inpcb: @F100610003EE258) --- SOCKET @ F100610003EEE00
--- TCP (inpcb: @F100610002AE0258) --- SOCKET @ F10061002AE0000
--- TCP (inpcb: @F100610002A6D258) --- SOCKET @ F10061002A6D000
--- TCP (inpcb: @F1006100034325B) --- SOCKET @ F10061000343300
--- TCP (inpcb: @F1006100043525B) --- SOCKET @ F10061000435000
--- TCP (inpcb: @F10061000437A58) --- SOCKET @ F10061000437800
--- TCP (inpcb: @F100610003F1258) --- SOCKET @ F100610003F1000
KDB(0)> sockinfo F100610003F0258 inpcb address of first inpcb in list above
--- TCPCB -----(@ F100610003F0360)----
seg_next....0F100610003F0360 seg_prev.......0F100610003F0360
 t_softerror... 00000000 t_state...... 00000001 (LISTEN)
t_timer.........00000000 (TCPT_REXMT)
t_timer.........00000000 (TCPT_PERSIST)
t_timer.........00000000 (TCPT_KEEP)
t_timer.........00000000 (TCPT_2MSL)
t_rxtshif......00000000 t_rxtcur......00000006 t_dupacks.....00000000
t_maxseg......00000200 t_force.......00000000
t_flags.......00000020 (RFC1323|COPYFLAGS)
t_oobflags....00000000 ()
t_template.....00000000 t_inpcb.......0F100610003F0258
t_jobc.........00000000 t_timestamp...6086EC01 snd_una.......00000000
snd nxt........00000000 snd_up.........00000000 snd_w11........00000000
snd_w2........00000000 iss...........00000000
snd wnd........000000000000000 rpv wnd........000000000000000
rcv nxt........00000000 rpv up........00000000 lrs...........00000000
snd wnd scale.00000000 rpv wnd scale.00000000 req scale_sent 00000000
req_scale_rcvd00000000 last_ack_sent.00000000 timestamp_rec.00000000
timestamp age.00000006 rpv adv.......00000000 snd_max.......00000000
snd_cwnd.....00000003FFC000 snd_ssthresh..000000003FFFC000
t_idle........ 00000006 t_rtt......... 00000000
t_srtt........ 00000000 t_rttvar...... 00000006
t_rttseq....... 00000000 max_sndwnd.... 0000000000000000
max_rxv........ 00000000 t_peermaxseg. 00000020
sack_data..... 0000000000000000 snd_in_pipe...
max_rcvd...... 0000000000000000
max_sndwnd.... 0000000000000000
snd_in_pipe... 00000000
snd_recover... 00000000
snd_high...... 00000000
snd_ecn_max... 00000000
snd_ecn_clear... 00000000
t_splice_with.@ 0000000000000000
t_splice_flags 00000000

-------- TCB -------- INPCB INFO ----(@ F1000610003F0258)----
next........ 0000000000000000 prev........ 0000000000000000
head........ 0000000003E4B780 faddr_6.....@ F1000610003F0278
iflowinfo... 00000000 fport....... 00000000
oflowinfo... 00000000 lport....... 00000000
t_splice_with.@ 0000000000000000
snd_in_pipe... 00000000
snd_recover... 00000000
snd_high...... 00000000
snd_ecn_max... 00000000
snd_ecn_clear... 00000000
t_splice_with.@ 0000000000000000
snd_in_pipe... 00000000
type........ 0000 (ACCEPTCONN|REUSEADDR)
proto....@ 0000000003E427A8
fdev_ch.@ F10006000CE0F480
rcv_t        00000000
hawat...@ 000000000000E000
mbcnt...@ 0000000000000000
mbmax...@ 0000000000003800
lowat...@ 00000000001000
mb......@ 0000000000000000
sel......@ 0000000000000000
events...... 0000
iodone..@ 00000000
ioargs..@ 0000000000000000
lastpkt..@ 0000000000000000
wakeup...@ 0000000000000000
lockwtg. FFFFFFFFEFFFF

MBUF LIST

rcv_t........@ 0000000000000000
hawat...@ 000000000000E000
mbcnt...@ 0000000000000000
mbmax...@ 0000000000003800
lowat...@ 00000000001000
mb......@ 0000000000000000
sel......@ 0000000000000000
events...... 0000
iodone..@ 00000000
ioargs..@ 0000000000000000
lastpkt..@ 0000000000000000
wakeup...@ 0000000000000000
lockwtg. FFFFFFFFEFFFF

MBUF LIST

tpcb....@ 0000000000000000
fdch_ch.0F10006000CE0F480
sec_info0000000000000000
qos........ 0000000000000000
gidlist.0000000000000000
private.0000000000000000
uid....@ 0000000000000000
bufsize. 0000000000000000
threadcnt0000000000000000

Chapter 36. Network subcommands 415
nextfree@0000000000000000
siguid.. 00000000 sigeid. 00000000 sigpriv.. 00000000
sndtime. 0000000000000000 sec 0000000000000000 usec
rcvtime. 0000000000000000 sec 0000000000000000 usec
saioq...0000000000000000 saioqhd.0000000000000000
accept.. FFFFFFFF FFFF FFFF FFFF frctime 00000000
isoflags 00000000 ()
rcvlen.. 0000000000000000 frcaback0000000000000000
frcassoc0000000000000000 frcabckt 0000000000000000
iodone.. 00000000 iodonefl 00000000 ()
ioarg...0000000000000000 refcnt.. 0000000000000000

proc/fd: 98/19
proc/fd: fd: 19
  SLOT NAME STATE PID PPID ADSPACE CL #THS
pvproc+018800 98/inetd ACTIVE 006206 0017056 0000002002D555 0 0001

KDB(0)>
**ndd subcommand**

**Purpose**

The **ndd** subcommand displays the network device driver statistics.

**Syntax**

```
ndd [-s | effectiveaddress | -n nddname]
```

**Parameters**

- **-s**
  
  Displays the list of all of the valid network device driver tables and gives the address of each **ndd** structure and the name of the corresponding network interface.

- **effectiveaddress**
  
  Specifies the effective address from which the **ndd** structure is read. Use symbols, hexadecimal values, or hexadecimal expressions to specify the address.

- **-n nddname**
  
  Indicates a network interface name is used to specify which **ndd** structure is to be read.

When it is used with an address or network interface name, the **ndd** subcommand displays a detailed description of the corresponding table. When it is used with the **-s** parameter, a list of valid network interfaces and the addresses of their **ndd** structures is printed. If no parameters are used, the **ndd** subcommand displays a detailed description of all of the valid network device driver tables.

**Aliases**

No aliases.

**Example**

The following is an example of how to use the **ndd** subcommand:

```
KDB(0)> ndd -s
--- NDD ADDR ---(@ F10010E00C69A030)---
  name.... ent1  alias.... en1
--- NDD ADDR ---(@ F10010E00C6AB030)---
  name.... ent0  alias.... en0
--- NDD ADDR ---(@ F10010E00BD64028)---
  name.... tok0  alias.... tr0

KDB(0)> ndd -n ent0
---- NDD INFO ----(@ F10010E00C6AB030)----
  name........... ent0  alias............ en0
  ndd_next........@F10010E00BD64028
  flags........... 0063091B
    (UP|BROADCAST|RUNNING|NOECHO|ALT ADDRS|64BIT|CHECKSUM_OFFLOAD|PSEG...
...
  ndd_open()..... 03D87690 ndd_close().... 03D876C0 ndd_output..... 03D876A8
  ndd_ctl()...... 03D876D8 ndd_stat().... 03D85A28 receive()...... 03D85A10
  ndd_refcnt..... 00000001  ndd_correlator...@F10010E00C6AB000
  ndd_mtu.......... 000005EA  ndd_mintu....... 0000003C
  ndd_addrlen..... 00000006  ndd_physaddr.... 000255AF36F2
  ndd_hdrlen..... 0000000E
  ndd_type........ 00000007 (802.3 Ethernet)
  ndd_demuxer...@0000000003D65BB8  ndd_nsdemux...@F10010F000340000
  ndd_demuxsource.. 00000000  ndd_specdemux...@F10010F000770000
  ndd_demux_lock... 0000000000000000  ndd_lock........ 0000000000000000
  ndd_trace.......0000000000000000  ndd_trace_arg...0000000000000000
  ndd_speclen..... 00000008C  ndd_specStats...@F10010E00C67BA0
  ndd_ipackets..... 000005E3  ndd_opackets..... 000060FA
  ndd_ierrors...... 00000000  ndd_oerrors...... 00000000
  ndd_ibytes....... 007C0235  ndd_obytes....... 00210113
  ndd_recvintr..... 00000287  ndd_xmitintr..... 00000002
  ndd_ipackets_drop 00000000  ndd_nobufs....... 00000000
```

Chapter 36. Network subcommands 417
KDB(0)> ndd_xmitque_max.. 00000004  ndd_xmitque_ovf.. 00000000
nsdbg subcommand

Purpose
The nsdbg subcommand displays the ns_alloc and free event records stored in the kernel.

Note: This functionality is only available if the ndd_event_tracing parameter is turned on by using the no command.

Syntax
nsdbg [-i starting_index] [-c display_count] [-n nddname[,nddname[,...]]]

Parameters
- **-i starting_index** Displays events starting with the event record specified with the starting_index parameter.
- **-c display_count** Displays only the events specified with the display_count parameter.
- **-n nddname** Displays the events associated with the network interface that have names specified with the nddname parameter.

If no parameters are specified, the nsdbg subcommand displays all event records stored in the kernel.

Aliases
No aliases.

Example
No example.
netstat subcommand

Purpose
The netstat subcommand symbolically displays the contents of various network-related data structures for active connections.

Syntax
netstat [-n] [-D] [-c] [-P] [-m | -ss | -u | -v] [ [ { -A | -a } | { -r | -C | -I Interface } ] ]
[-f AddressFamily] [ -p Protocol ] [-Zc | -Zi | -Zm | -Zs ] [ Interval ] [ System ]

Parameters
-n  Shows network addresses as numbers. When the -n flag is not specified, the netstat command interprets addresses where possible and displays them symbolically. This flag can be used with any of the display formats.
-D  Shows the number of packets received, transmitted, and dropped in the communications subsystem.
-c  Shows the statistics of the Network Buffer Cache.
-P  Shows the statistics of the Data Link Provider Interface (DLPI).
-m  Shows statistics recorded by the memory management routines.
-s  Shows statistics for each protocol.
-ss  Displays all of the non-zero protocol statistics and provides a concise display.
-u  Displays information about domain sockets.
-v  Shows statistics for CDLI-based communications adapters. This flag causes the netstat command to run the statistics commands for the entstat subcommand, the tokstat subcommand, and the fddistat subcommand. No flags are issued to these device driver commands.
-A  Shows the address of any protocol control blocks associated with the sockets. This flag acts with the default display and is used for debugging purposes.
-a  Shows the state of all of the sockets. Without this flag, sockets used by server processes are not shown.
-r  Shows the routing tables. Shows routing statistics when it is used with the -s flag.
-C  Shows the routing tables, including the user-configured costs and current costs of each route.
-i  Shows the state of all configured interfaces.
-I Interface  Shows the state of all of the configured interfaces specified by the Interface variable.
-f AddressFamily  Limits reports of statistics or address control blocks to those items specified by the AddressFamily variable. The following address families are recognized:
- inet  – Indicates the AF_INET address family
- inet6  – Indicates the AF_INET6 address family
- ns  – Indicates the AF_NS address family
- unix  – Indicates the AF_UNIX address family
-p Protocol  Shows statistics about the value specified for the Protocol variable, which is either a name for a protocol or an alias for it. Protocol names and aliases are listed in the /etc/protocols file. A null response means that there are no numbers to report. The program report of the value specified for the Protocol variable is unknown if there is no statistics routine for it.
-Zc  Clears network buffer cache statistics.
-Zi  Clears interface statistics.
-Zm  Clears network memory allocator statistics.
-Zs  Clears protocol statistics. To clear statistics for a specific protocol, use -p Protocol. For example, to clear the TCP statistics, type the following on the command line:
netstat -Zs -p tcp

Aliases
No aliases.
Example

The following is an example of how to use the `netstat` subcommand:

```
<0>netstat -r
```

Route Tree for Protocol Family 2 (Internet):  
default advantis.in.ibm.c UGc 0 0 en0 - -  
freezer.austin.i 9.184.199.232 UGHMW 0 1 en0 - 1  
9.184.192/21 shakti.in.ibm.com U 20 40546 en0 - -  
mqet2.in.ibm.com 9.184.199.12 UGHMW 0 958 en0 - 1  
127/8 localhost U 2 249 lo0 - -  

Route Tree for Protocol Family 24 (Internet v6):  
::1 ::1 UH 0 0 lo0 16896 -  

-----------------------------------------------------------------------------

Chapter 36. Network subcommands
route subcommand

**Purpose**
The `route` subcommand displays the `route` structure at a given address.

**Syntax**
```
route effectiveaddress
```

**Parameters**
- `effectiveaddress` Specifies the effective address of the `route` structure to display.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `route` subcommand:
```
# netstat -f inet -n -A
Active Internet connections
PCB/ADDR Proto Recv-Q Send-Q Local Address Foreign Address (state)
715a45e8 tcp4 0 0 9.53.85.113.23 9.53.85.114.50921 ESTABLISHED
```

```
# Debugger entered via keyboard.
.KDB(0)> tpcb 715a45e8 //tpcb address from PCB/ADDR column in netstat
--- TCP/PCB ---(@ 715A45E8)---
   seg_next...... 715A45E8 seg_prev..... 715A45E8
   t_softerror... 00000000 t_state...... 00000004 (ESTABLISHED)
   t_timer........ 00000000 (TCPT_REXMT)
   t_timer........ 00000000 (TCPT_PERSIST)
   t_timer........ 000037D7 (TCPT_KEEP)
   t_rtxshift..... 00000000 t_rxtCur...... 00000003 t_dupacks..... 00000000
   t_maxseg...... 000005B4 t_force....... 00000000
   t_flags........ 00080000 ()
   t_oobflags..... 00000000 ()
   t_iobc......... 00000000 t_template.... 715A4610 t_inpcb....... 715A4544
   t_ratio........ 00000000 t_template.... 715A4610 t_inpcb....... 715A4544
   t_nxt......... 00000000 t_prev........ 00000000
   t_head........ 02576600
   f_addr_6...... @ 715A4558
   f_port........ 0000C6E9
   f_type......... 00000001
   l_addr_6...... @ 715A4570
   l_port......... 0000017
   p_ppcb........ 715A45E8 route_6...... @ 715A4588 ifa............ 00000000
   flags........ 00000000 proto........ 00000000 tos......... 00000000
```

KDB(0)> tcb 715A4544 //tcb address from the t_inpcb field
--- TCB ---(@ 715A4544)---
```
next........ 00000000 prev........ 00000000 head........ 02576600
i_flowinfo... 00000000 faddr_6... @ 715A4558 fport........ 0000C6E9
fatype....... 00000001 oflowinfo... 00000000 laddr_6... @ 715A4570
lport........ 00000017 latype....... 00000001 socket....... 715A4400
pppcb........ 715A45E8 route_6... @ 715A4588 ifa............ 00000000
flags........ 00000040 proto........ 00000000 tos......... 00000000
```
Chapter 36. Network subcommands 423

KDB(0)> route 715A4588 //route address from the route_6 field

Destination.. 9.53.85.114

........rtentry@ 715AEE00........

rt_nodes[0]......

  rn_mklst @.. 701FA2E0
    rm_b.......... FFFFFFFC7 rm_unused......
    rm_flags...... 00000005 rm_mklst...... 00000000
    rmu_mask...... 701F51B0
    mask.......... 255.255.255.0
    rm.refs...... 00000000

  rn_p @....... 715AED18
  rn_b.......... FFFFFFFC7 rm_bmask..... 0000
  rn_flags...... 0000000D (NORMAL|ACTIVE|DUP)
  rn_key........ 9.53.85.0/24

  rn_dupedkey @ 00000000

rt_nodes[1]......

  rn_mklst @.. 00000000
  rn_p @....... 7095D11B
  rn_b.......... 00000024 rn_bmask..... 0008
  rn_flags...... 00000004 (ACTIVE)
  rn_off........ 00000004
  rn_l @....... 701FFC2C rn_r @....... 7095D518

gateway...... 9.53.85.113
rt_redistctime 00000000 rt_refcnt...... 00000003
rt_flags...... 00000001 (UP)
ifnet @...... 334A6000 ifaddr @...... 701F5100
rt_genmask @. 00000000 rt_llinfo @.. 00000000
rt_rmx (rt_metrics):
    locks ... 00000000 mtu ..... 00000000 hopcount. 00000000
    expire .. 401FDFCB recvpipe. 00000000 sendpipe. 00000000
    ssthresh. 00000000 rtt ..... 00000000 rttvar .. 00000000
    pksent... 00000031
rt_gwroute @. 00000000 rt_idle...... 00000000
ipRouteAge... 00000000 rt_proto @... 00000000
gidstruct @. 00000000 rt_lock...... 00000000
rt_intr....... 00000003 rt_duplist @. 00000000
rt_lu @...... 00000000 rt_timer..... 00000000
rt_cost_config 00000000

KDB(0)>
rtentry subcommand

Purpose
The `rtentry` subcommand displays the `rtentry` structure at a given address.

Syntax
```
rtentry effectiveaddress
```

Parameters
```
effectiveaddress       Specifies the effective address of the rtentry structure to display.
```

Aliases
No aliases.

Example
The following is an example of how to use the `rtentry` subcommand:
```
# netstat -f inet -r -A -n
Routing tables
Address   Destination   Gateway   Flags   Refs   Use   If      PMTU   Exp   Groups

Route tree for Protocol Family 2 (Internet):
701fcc44 (32) 7095d118 : 701fcc5c mk = 70a9f080 {0, (0)}
7095d118 (33) 715aee18 : 7095d100
715aee18 (36) 701fcc2c : 7095d518
701fcc2c 70a5b100 default 9.53.85.1 UGc 0 0 en0 - -
    mask (0) mk = 70a9f080 {0, (0)}
7095d518 (42) 7095d500 : 727bad18
7095d500 9.0.7.1 9.53.85.1 UGHW 0 628 en0 1500 1
727bad18 (43) 727bad00 : 715aed18
727bad00 9.41.85.44 9.53.85.1 UGHW 0 2 en0 - 1
715aed18 (56) 7095d218 : 715aed00 mk = 701fa2e0 {56, (0) 0 ffff ff00}
7095d218 (57) 715ae00 : 7095d200
715ae00 9.53.85.0 9.53.85.113 UHSb 0 0 en0 - - =>
    715ae00 9.53.85/24 9.53.85.113 U 4 49 en0 - -
    mask (0) ffff ff00 mk = 701fa2e0 {56, (0) 0 ffff ff00}
7095d200 9.53.85.113 127.0.0.1 UGHs 0 1195 lo0 - -
715aed00 9.53.85.255 9.53.85.113 UHSb 0 1 en0 - -
7095d100 127/8 127.0.0.1 U 2 831 lo0 - -
    mask (0) 0 ff00
701fcc5c # Debugger entered via keyboard.
./waitproc_find_run_queue+000048    ori r3,r8,0  <-00000000> r3=pnda,r8=0
KDB(0)> rtentry 727bad00 //rtentry address from Routing Address column in netstat

............rtentry@ 727BAD00............
```

```
rt_nodes[0]......
    rn_mklist 0.. 00000000
    rn_p @........ 727BAD18
    rn_b......... FFFFFFFF rn_bmask..... 0000
    rn_flags..... 00000004 (ACTIVE)
    rn_key....... 9.41.85.44
    rn_dupedkey @ 00000000

rt_nodes[1]......
    rn_mklist 0.. 00000000
    rn_p @........ 70950518
    rn_b......... 0000002B rn_bmask..... 0010
```

Chapter 36. Network subcommands  425
rn_flags..... 00000004 (ACTIVE)
rn_off....... 00000005
rn_l @....... 727BAD00 rn_r @....... 715AE000

gateway...... 9.53.85.1
rt_redistime 00000000 rt_refcnt.... 00000000
rt_flags..... 00020007 (UP|GATEWAY|HOST|CLONED)
ifnet @...... 33460000 ifaddr @...... 701F5100
rt_genmask @. 00000000 rt_linfo @. 00000000
rt_rmx (rt_metrics):
   locks.... 00000000 mtu ..... 00000000 hopcount. 00000000
   expire .. 401FE02A recvpipe. 00000000 sendpipe. 00000000
   ssthresh. 00000000 rtt ..... 00000000 rttvar .. 00000000
   pktsent... 00000002
rt_gwroute @. 715AE000 rt_idle...... 00000000
ipRouteAge... 00000000 rt_proto @... 7095F4A0
gidstruct @.. 70958800 rt_lock...... 00000000
rt_intr...... 00000000 rt_duplist @. 00000000
rt_lu @...... 00000000 rt_timer..... 00000000
rt_cost_config 00000000

KDB(0)>
**rxnode subcommand**

**Purpose**
The `rxnode` subcommand displays information about the *radix_node* structure at a specified address.

**Syntax**
```
rxnode effectiveaddress
```

**Parameters**
```
effectiveaddress   Specifies the effective address of the *radix_node* structure.
```

After displaying the *radix_node* structure, the subcommand presents a menu for interactive traversal of the *radix_node* tree. If the *radix_node* is an intermediate node of the tree, the traversal can follow the parent, left, or right nodes. If the displayed *radix_node* is a leaf node, the traversal can only follow the parent node.

**Aliases**
No aliases.

**Example**
The following is an example of how to use the `rxnode` subcommand:

```
# netstat -f inet -r -A -n
Routing tables
Address         Destination   Gateway   Flags   Refs   Use   If   PMTU   Exp   Groups
Route tree for Protocol Family 2 (Internet):
701fcc44 (32) 7095d118   : 701fcc5c mk = 70a9f080 {} {}, }
7095d118 (33) 715ae18    : 7095d100
715ae18 (36) 701fcc2c    : 7095d18
701fcc2c 70a5b100 default: 9.53.85.1 UGc 0 0 en0 - -
mak (0) mk = 70a9f080 {} {}, }
7095d18 (42) 7095d500   : 715aed18
7095d500 9.0.7.1   9.53.85.1 UGHW 0 1121 en0 - 2
715aed18 (56) 7095d218   : 715aed00 mk = 701fa2e0 {}(56), {}, 0 ffff ff00 }
7095d218 (57) 715ae00    : 7095d200
715ae00 9.53.85.0   9.53.85.113 UHSb 0 0 en0 - - =>
715ae00 9.53.85.24   9.53.85.113 U 3 80 en0 - -
mak (0) 0 ffff ff00 mk = 701fa2e0 {}(56), {}, 0 ffff ff00 }
7095d200 9.53.85.113 127.0.0.1 UGHW 2 2221 lo0 - -
715aed00 9.53.85.255 9.53.85.113 UHSb 0 1 en0 - -
7095d100 127/0 127.0.0.1 U 2 1469 lo0 - -
mak (0) 0 ff00
701fcc5c # Debugger entered via keyboard.
....wtproc+00000E8 ori r3,r31,0 <003F3780> r3=0,r31=ppda
KDB(0)> rtentry 7095d200 //rtentry address from Routing Address column in netstat

........rtentry@ 70950200........

rt_nodes[0]......
```

```

rn_mklist 0.. 00000000
rn_p 0...... 7095d218
rn_b........ FFFFFFFFFFFF rn_bmask..... 0000
rn_flags..... 00000004 (ACTIVE)
rn_key....... 9.53.85.113
rn_dupedkey @ 00000000
```
rt_nodes[1]......

```
    rn_mklist 0.. 00000000
    rn_p 0....... 715AED18
    rn_b........ 0000039  rn_bmask.... 0040
    rn_flags..... 00000004 (ACTIVE)
    rn_off....... 00000007
    rn_l 0....... 715AEF00  rn_r 0....... 7095D200
    gateway.... 0.. 127.0.0.1
    rt_redisctime 00000000  rt_refcnt.... 00000002
    rt_flags..... 00000807   (UP|GATEWAY|HOST|STATIC)
    ifnet 0...... 011EDB70  ifaddr 0...... 7095C000
    rt_genmask 0.. 00000000  rt_llinfo 0.. 00000000
    ifnet........
    ifaddr........
    rt_genmask........
    rt_llinfo........
    rt_mrx (rt_metrics):
        locks ... 00000000  mtu .... 00000000
        expire .. 00000007  recvpipe. 00000000
        ssthresh. 00000000  rtt ..... 00000000
        pktsent... 00000000
    rt_gwroute 0. 7095D100  rt_idle...... 00000000
    ipRouteAge... 00000000  rt_proto 0... 7095F160
    gidstruct 0.. 00000000  rt_lock..... 00000000
    rt_intr....... 00000009  rt_duplist 0. 00000000
    rt_lu 0....... 00000000  rt_timer..... 00000000
    rt_cost_config 00000000
```

KDB(0)> rxnode 715AEF00 //radix node address from rn_l; can also use rn_r or rn_p

```
    rn_mklist 0.. 00000000
    rn_p 0....... 7095D218
    rn_b........ FFFFFFFF  rn_bmask.... 0000
    rn_flags..... 00000000 (NORMAL|ACTIVE|DUP)
    rn_key........ 9.53.85.0
    rn_dupedkey 0 715AEE00
    Traverse radix_node tree:
       parent - 1  quit - 0
    Enter Choice : 1

    rn_mklist 0.. 00000000
    rn_p 0....... 715AED18
    rn_b........ 00000039  rn_bmask.... 0040
    rn_flags..... 00000004 (ACTIVE)
    rn_off....... 00000007
    rn_l 0....... 715AEF00  rn_r 0....... 7095D200
    Traverse radix_node tree:
       parent - 1  rn_r - 2  rn_l - 3  quit - 0
    Enter Choice : 2

    rn_mklist 0.. 00000000
    rn_p 0....... 7095D218
    rn_b........ FFFFFFFF  rn_bmask.... 0000
    rn_flags..... 00000004 (ACTIVE)
    rn_key........ 9.53.85.113
    rn_dupedkey 0 000000000
    Traverse radix_node tree:
       parent - 1  quit - 0
    Enter Choice : 0
```

KDB(0)>

---

428  KDB Kernel debugger and kdb command
tcpdbg subcommand

Purpose
The tcpdbg subcommand displays the tcp_debug structures. The amount of information displayed depends on the socket trace level set.

Syntax
```
tcpdbg [-i index] [-c count] | [-a address] | [-s]
```

Parameters
- **tcpdbg**
  Displays the last 10 tcp_debug entries at the current index.
- **-c count**
  Defines how many entries to display. The default value is 10. The count specified must be lower than the value defined by tcp_ndebug variable.
- **-i index**
  Displays the last count (10 if count is not specified by the -c parameter) tcp_debug entries starting from the one at the index specified in the index parameter.
- **-a address**
  Displays the tcp_debug structures at the address specified with the address parameter.
- **-s**
  Displays the socket details associated with the tcp_debug entries. It is available with -a, -i and -c options only if the trace level >= normal.

Aliases
No aliases.

Example
- The following example displays one structure of tcp_debug at index 31 for detail trace level.
  ```c
  KDB(0)> tcpdbg -i 31 -c 1
  ---- TCP_DEBUG ----(0 70293358)----
  (tcp_debx= 31)
  act.....0001 (OUTPUT) ostate.....0004 (ESTABLISHED)
  ADDRESS family.....02 (AF_INET)
  TRACE level........0007 (SO_TRC_DTL)
  TCPIP hdr : next........000100034 prev........000004000
  pr.........00000001 
  len.........0020
  src.........172.16.101.137 dst.........172.16.101.137
  sport......8007 dport......0017
  seq.........4E015C41 ack......C8075EF0 off.......00000008
  flags......00000010 (TH_ACK)
  win.........00000006 sum.......00001449 urp.......0000
  ---- TCPCB ----(0 72409E00)----
  seg_next...............072409E00 seg_prev...............072409E00
  t_softerror... 00000000 t_state..... 00000004 (ESTABLISHED)
  t_timer........ 00000000 (TCPT_RMTX)
  t_timer........ 00000000 (TCPT_PERSIST)
  t_timer........ 00000000 (TCPT_KEEP)
  t_timer........ 00000000 (TCPT_2MSL)
  t_rxtshift.... 00000000 t_rxtcur...... 00000003 t_dupacks..... 00000000
  t_maxseg.... 00000000 t_force....... 00000000
  t_flags......... 000003E0 (RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS|COPYFLAGS)
  t_oobflags..... 00000000
  t_template...... 00000000 t_inpcb............... 072409E00
  t_iobc........ 00000000 t_timestamp.... 08915C01 snd_una.... 4E015C41
  snd_nxt........ 4E015C41 snd_up........ 4E015C41 snd_w11..... C8075669
  snd_w12...... 4E015C41 iss............ 4E015AE7
  snd_wnd............ 0020E60rcv_wnd............ 0020E60
  rcv_nxt........ C8075EF0 rcv_up........ C8075667 irs........... C805EDFA
  ```
The following example displays one structure of tcp_debug at index 29 with socket information for detail trace level.

KDB(0)> tcpdbg -i 29 -c 1 -s

---- TCP_DEBUG ----(@ 70293008)----
(tcp_debx= 29)

act.....0000 (INPUT) ostate.....0004 (ESTABLISHED)
ADDRESS family.....02 (AF_INET)
TRACE level........0007 (SO_TRC_DTL)
TCPIP hdr : next.........@00000000 prev............@00000000
pr............00000006
len............0887
crc............172.16.101.137 dst............172.16.101.137
src............0017 dport............8007
seq............C8075669 ack.....4E015C41
off............00000008
flags......00000018 (TH_PUSH|TH_ACK)
win............00008398 sum............000017A3

---- TCPCB ----(@ 72409E00)----
seg_next...............@72409E00 seg_prev...............@72409E00
t_softerror...00000000 t_state......00040004 (ESTABLISHED)
t_timer............00000000 (TCPT_REXMT)
t_timer............00000000 (TCPT_PERSIST)
t_timer............00000000 (TCPT_KEEP)
t_timer............00000000 (TCPT_2MSL)
t_rxtshift....00000000 t_rxtcur......00000003 t_dupacks....00000000
t_maxseg....000041CC t_force............00000000
t_flags....000003E0 (RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS|COPYFLAGS)
t_oobflags....00000000
_t_template......00000000 t_inpcb......000000000072409D54
_t_iobc............00000000 t_timestamp...08915C01 t_snd stable...4E015C41
_t_snd_nxt......4E015C41 t_snd_up......4E015C41 t_snd_w1......C8075669
t_snd_w2......4E015C41 t_snd_w2......4E015C41
snd_wnd............000020E60rcv_wnd............000020E60
rcv_nxt............C8075EF0 rcv_up............C8075667 t_sms............C805ED8A
snd_wnd_scale.00000002 rcv_wnd_scale.00000002 req_scale_sent.00000002
req_scale_rcvd.00000002 last_ack_sent.00075E00 timestamp_rec.00000000
timestamp_age.00000004 rcv_adv....4E096D50 snd_max....4E015C41
snd_cwnd............00000002 max_sndwnd.......000020E60
snd_ecn_max....4E015C41
snd_ecn_clear....4E015C42
t_peermaxseg..000041CC snd_in_pipe...00000000
sack_data............00000000 t_snd_recover..00000000
snd_high.......4E015C41 snd_ecn_max...4E015C41 snd_ecn_clear.4E015C42
t_splice_with............00000000 t_splice_flags 00000000

---- SOCKET INFO ----(@ 72409E00)----
type........0001 (STREAM)
opts........0101 (DEBUG|OOBINLINE)
linger.......0000 state......0082 (ISCONNECTED|PRIV)
pcb............072409D54 proto............02681700
lock............07240F2E0 head............00000000
q0............00000000 q1............00000000
q2len........00000000
The following example displays one structure of tcp_debug at address 70293008 for detail trace level.

```
KDB(0)> tcpdbg -a 70293008

---- TCP_DEBUG ----(@ 70293008)----

act.....000 (INPUT) ostate.....0004 (ESTABLISHED)
ADDRESS family.....02 (AF_INET)
TRACE level........0007 (SO_TRC_DTL)
TCP/IP hdr : next.........@00000000 prev........@00000000
pr.........00000000 len.........0887
src........172.16.101.137 dst.........172.16.101.137
sport.......0017 dport.......8007
seq.........C8075669 ack.......4E015C41 off.......00000008
flags.......00000018 (TH_PUSH|TH_ACK)
win.........00008398 sum.......000017A3 urp.......0000

---- TCPCB ----(@ 72409E00)----

seg_next........@72409E00 seg_prev........@72409E00
 t_softerror... 00000000 t_state..... 00000004 (ESTABLISHED)
 t_timer........ 00000000 (TCPT_PERSIST)
 t_timer........ 00000000 (TCPT_PERSIST)
 t_timer........ 00000000 (TCPT_2MSL)
 t_rxtshift.... 00000000 t_rtxcur...... 00000003 t_dupacks.... 00000000
 t_mseg....... 000001CC t_force....... 00000000
 t_flags....... 000003E0 (RFC1323|SENT_WS|RCVD_WS|SENT_TS|RCVD_TS|COPYFLAGS)
 t_oobflags.... 00000000
 t_template........@00000000
 t_inpcb.........@00000000 t_inpcb......@072409D54
 t_iobc........ 00000000 t_timestamp... 08915C01 snd_una...... 4E015C41
 snd_mxt........ 4E015C41 snd_up........ 4E015C41 snd_w1...... C8075669
 snd_w2........ 4E015C41 iss........ 4E015AE7
 snd_wnd........ 00020E60rcv_wnd.............. 00020E60
 rccv_mxt........ C80756E0 rccv_up........ C8075667 rss........ C805ED8A
 snd_wnd_scale. 00000002 rccv_wnd_scale. 00000002 req_scale_sent 00000002
 req_scale_rcvd 00000002 last_ack_sent. C80756E0 timestamp_rec. 204CF31A
 timestamp_age.. 00000040 rcc_adv........ C8096050 snd_max........ 4E015C41
 snd_cum........ 0003FFFC snd_ssthresh..... 3FFFC000
 t_idle......... 00000004 t_rtt......... 00000000 t_rtseq...... 4E015C3F
```

etc...

- The following example displays one structure of tcp_debug at address 70293008 for detail trace level.
The following example displays one structure of tcp_debug at index 88 with socket information for normal trace level.

```
KDB(0)> tcpdbg -i 88 -c 1 -s

---- TCP_DEBUG ----(0 702991C0)----
tcp_debx= 88

act.....0000 (INPUT) ostate.....0004 (ESTABLISHED)
ADDRESS family.....02 (AF_INET)
TRACE level........0003 (SO_TRC_NORM)
TCPIP hdr : next...........0000000000 prev...........00000000
pr...........00000006 len........0001
src.........172.16.101.137 dst.........172.16.101.137
sport......0017 dport......8007
seq.........C807086D ack.....4E015BF1 off........00000008
flags........00000018 (TH_PUSH|TH_ACK)
win...........000008398 sum........0000FF71 urp......0000

---- TCP PCB ----(0 72409E00)----

---- SOCKET INFO ----(0 72409C00)----
```
Chapter 37. Workload Manager (WLM) subcommands

The subcommands in this category support the WLM functions. These subcommands include the following:

- `cla`
- `rules`
- `bdev`
- `bqueue`
cla subcommand

Purpose
The cla subcommand displays Workload Manager (WLM) class statistics and configuration information.

Syntax
cla * [select#]
cla [classid]

Parameters
* Displays the menu if the select# parameter is not specified.
select# Displays the class statistics for the selected number.
1 – CPU for all classes
2 – Mem for all classes
3 – Mem for superclasses
4 – CPU for all classes
5 – Mem for one superclass
6 – BIO use for all classes
7 – BIO use for active classes
8 – BIO use, per-disk, for all classes
9 – Totals for all classes and all resources
classid Displays configuration information for the specified class identifier.

Aliases
class

Example
The following is an example of how to use the cla subcommand completed by using the menu:

KDB(0)> cla *
WLM CLASSes
Select the criteria to display by:
1) CPU use
2) MEM use
3) MEM use over superclasses
4) Superclasses only
5) MEM use inside a superclass
6) BIO use
7) BIO use (show active classes for all disks)
8) BIO use (show classes for all disks)
9) Total Resources
Enter your choice: 1
(wlm is ON)
TIER US MIN SHA SMAX HMNX DES RAP URApH URApI URApL PRI NT TB TOTALTB
[0]: Unclassified 0 0 0 -1 100 100 100 0 0 194 10 0 0x00000000 0x00000000
[64]: Unmanaged 0 0 0 -1 100 100 100 0 0 194 10 0 0x00000000 0x00000000
[128]: Default 0 0 0 -1 100 100 100 0 0 194 0 0 0x00000000 0x00000000
[129]: Default.Default 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000
[130]: Default.Shared 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000
[192]: Shared 0 0 0 -1 100 100 100 0 0 194 0 0 0x00000000 0x00000000
[193]: Shared.Default 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000
[194]: Shared.Shared 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000
[256]: System 0 1 0 -1 100 100 100 0 0 194 0 0 0x000043A8 0x00000000
[257]: System.Default 0 1 0 -1 100 100 100 0 0 97 0 1 0x000043A8 0x00001085
[258]: System.Shared 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000
[320]: Test1 0 0 0 -1 100 100 100 0 0 194 0 0 0x00000000 0x00000000
[321]: Test1.Default 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000
[322]: Test1.Shared 0 0 0 -1 100 100 100 0 0 97 0 1 0x00000000 0x00000000

Display configuration for class 256
KDB(0)> cla 256
System (valid) wlm is UseClassif CpuAcct CpuRegul MemAcct MemRegul BioAcct BioRegul TotalCpuAcct TotalCpuRegul TotalDiskioAcct TotalDiskioRegul TotalConnectAcct TotalConnectRegul TotalProcAcct TotalProcRegul TotalThrAcct TotalThrRegul

434 KDB Kernel debugger and kdb command
### Display CPU statistics for all classes

KDB(0)> cl a + 1

<table>
<thead>
<tr>
<th>KDB(0)&gt;</th>
<th>cl a + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>wlm is ON</td>
<td>TIER</td>
</tr>
<tr>
<td>0:</td>
<td></td>
</tr>
<tr>
<td>64:</td>
<td></td>
</tr>
<tr>
<td>128:</td>
<td></td>
</tr>
<tr>
<td>129:</td>
<td></td>
</tr>
<tr>
<td>130:</td>
<td></td>
</tr>
<tr>
<td>139:</td>
<td></td>
</tr>
<tr>
<td>149:</td>
<td></td>
</tr>
<tr>
<td>256:</td>
<td></td>
</tr>
</tbody>
</table>

**Column Description**

- **First column**: The first column is the class ID. The superclass IDs are a multiple of 64, with subclass IDs following numerically.
- **Second column**: The second column is the class name. Subclass names have the format `<Supername.Subname>`.
- **TIER**: The tier number for the class. The tier for a superclass is the superclass. The tier for a subclass is the subclass.
- **%MIN**: The current consumption percentage for the class for the resource being displayed.
- **SHA**: The user-defined minimum limit for the class (default is 0).
- **SMAX**: The user-defined soft maximum limit for the class. (default is 100)
- **HMAX**: The user-defined hard maximum limit for the class. (default is 100)
- **DES**: The desired or target percentage for the class.
- **RAP**: The Resource Access Priority for the class. This is a value in the range [-100..100].
- **URAP**: The highest URAP value the class can attain.
- **URAPL**: The current URAP value for the class.
- **SBAP**: The lowest URAP value the class can attain.
- **NT**: The class priority.
- **TB**: Timebase units of consumption for the last second.
- **TOTALTB**: Decayed total timebase units of consumption.
bdev subcommand

Purpose
The bdev subcommand displays Workload Manager (WLM) I/O statistics for block devices.

Syntax
bdev [a] [c] [s] * -d major minor | effectiveaddress

Parameters
a Displays detailed (all) I/O statistics.
c Displays I/O statistics for each class.
s Displays I/O statistics for each device. This is the default.
* Displays I/O statistics for all managed devices.
-d Displays I/O statistics for a device specified by the major and minor numbers.
major Specifies the major number. This is a hexadecimal value.
minor Specifies the minor number. This is a hexadecimal value.
effectiveaddress Specifies the major number. This is a hexadecimal value, or hexadecimal expressions can be used to specify the address.

Aliases
wlm_bdev

Example
The following is an example of how to use the bdev subcommand:

Display summary statistics for all devices

KDB(0)> bdev *
33507000: " dev: 14,0 in_queue: 0 classes: 0 rq/s: 0 act: 0
33459000: " dev: 14,1 in_queue: 0 classes: 0 rq/s: 0 act: 0
334B0000: + dev: 14,2 in_queue: 2 classes: 7 rq/s: 157 act: 100

Description of output (above)

Column Description
1 eaddr of bdev control block
2 status of device
 -" = uregulated
 "~" = no activity
 "+" = active
3 dev: device major, minor number
4 in_queue: number of requests enqueued
5 classes: number of active classes for the device
6 rq/s: number of requests per second for the device
7 act: the percent active for the device

Display statistics for device with major # 14 and minor # 2

KDB(0)> bdev s -d 14 2
334B0000: + dev: 14,2 in_queue: 2 classes: 7 rq/s: 157 act: 100
flags 0x00000000 lock 0x00000000
next 00000000 nb_cntr 343344
ctrl 334B00F0 regul 334C14F0
delayed 32AEE000 in_use 32
ev_want_free 0xFFFFFFFF wbd_act_centr 7
wbd_in_queue 2 wbd_max_queue 6
dskStat 32A05274 prev_dk_time 919418
&current 334B00CC &info.wbd_last 334B0024
&info.wbd_max 334B003C &info.wbd_av 334B0054
&info.wbd_total 334B0070
Type RTHR WTHR RQSTS QUEUE STRVD ACTVT
   current 0 672 42 11 0 0
wbd_last 0 2512 157 36 0 100
wbd_max 9280 12192 1172 145 1 100
wbd_av 0 2441 152 33 0 100

436 KDB Kernel debugger and kdb command
Examples for B10 statistics using cla command

KDB(0) cla * 6

(wlm is ON)

<table>
<thead>
<tr>
<th>TIER</th>
<th>% MIN</th>
<th>SHA</th>
<th>SMAX</th>
<th>HMAX</th>
<th>DES</th>
<th>RAP</th>
<th>USRAP</th>
<th>URAP</th>
<th>URAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>511</td>
</tr>
<tr>
<td>64:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>193:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>257:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KDB(0) cla * 7

(wlm is ON)

<table>
<thead>
<tr>
<th>TIER</th>
<th>% MIN</th>
<th>SHA</th>
<th>SMAX</th>
<th>HMAX</th>
<th>DES</th>
<th>RAP</th>
<th>USRAP</th>
<th>URAP</th>
<th>URAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>320:</td>
<td>25</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>511</td>
</tr>
<tr>
<td>321:</td>
<td>25</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>511</td>
</tr>
<tr>
<td>322:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>323:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>384:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>386:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>448:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>449:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>513:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>514:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KDB(0) cla * 8

(wlm is ON)

<table>
<thead>
<tr>
<th>TIER</th>
<th>% MIN</th>
<th>SHA</th>
<th>SMAX</th>
<th>HMAX</th>
<th>DES</th>
<th>RAP</th>
<th>USRAP</th>
<th>URAP</th>
<th>URAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>511</td>
</tr>
<tr>
<td>64:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>193:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>257:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 37. Workload Manager (WLM) subcommands 437
### KDB Kernel debugger and kdb command

| Test1.Default | 0 0 0 -1 100 100 100 100 0 0 255 |
| Test2.Default | 0 0 0 -1 100 100 100 100 0 0 255 |
| Test3.Default | 0 0 0 -1 100 100 100 100 0 0 255 |

### Test1.Shared

| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |

### Test2.Shared

| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |

### Test3.Shared

| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |

### Test4.Default

| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |

### Test4.Shared

| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |
| 0 0 0 -1 100 100 100 100 0 0 255 |
bqueue subcommand

Purpose
The bqueue subcommand displays a queue of delayed Workload Manager I/O requests.

Syntax
bqueue effectiveaddress

Parameters
effectiveaddress Specifies the address of the head of the queue. This address can be obtained from the delay field from the output of the bdev subcommand.

Aliases
wlm_bq

Example
The following is an example of how to use the bqueue subcommand:
KDB(0)> bqueue 32AEE000
BUF urap next time
0000000032AEE000 120 0000000032AE8A00 0xA95AE221 (tod+23 ms)
0000000032AE8A00 247 0000000000000000 0xA95AE21E (tod+20 ms)

Description of output
BUF This is the address of the buf struct in the queue
urap The urap (class priority) of the requesting class
next The next buf in the queue
time The expiration time for the request (time to flush)
rules subcommand

Purpose
The rules subcommand displays the currently-loaded Workload Manager (WLM) assignment rules.

Syntax
rules

Parameters
There are no parameters. The output is in the following format:

  <address>: <classid> ("<classname>") <uidlist> <gidlist> <filelist>

where <filelist> is in the following format:

  (<device>.<inode>.<generation>)

A dash (−) means that the list is empty (unspecified).

Aliases
rule

Example
The following is an example of how to use the rules subcommand:

KDB(0)> rules
KERN_heap+ABA0C00:  320 ("Test1") 1220 - - -
KERN_heap+ABA0C58:  384 ("Test2") 1219 - - -
KERN_heap+ABA0CB0:  448 ("Test3") 1218 - - -
KERN_heap+ABA0D08:  512 ("Test4") - 1 - -
KERN_heap+ABA0D60:  576 ("Test5") - - (8000000A00000005.00001884.9F1CA805) - -
KERN_heap+ABA0DC8:  256 ("System") 0 - - -
KERN_heap+ABA0E20:  128 ("Default") - - - -
-----------------------------------------------
KERN_heap+ADAF000:  515 ("Test4.sub1") - - (8000000A00000005.0000187D.9F19E74) - -
KERN_heap+ADAF068:  516 ("Test4.sub2") - - (8000000A00000005.00001895.9F19E74) - -
-----------------------------------------------
KERN_heap+ABA2100:  579 ("Test5.sub1") - - (8000000A00000005.0000187D.9F19E74) - -
KERN_heap+ABA2168:  580 ("Test5.sub2") - - (8000000A00000005.00001895.9F19E74) - -
-----------------------------------------------
Appendix A. kdb Command

Purpose
Allows examining of a system dump or a running kernel.

Syntax

```
kubectl
```

```
```

```
```

Description

The kdb command is an interactive utility for examining an operating system image or the running kernel. The kdb command interprets and formats control structures in the system and provides miscellaneous functions for examining a dump.

The SystemImageFile parameter specifies the file that contains the system image. The value can indicate a system dump, the name of a dump device, or the /dev/pmem special file. The default SystemImageFile is /dev/pmem.

The KernelFile parameter specifies the AIX kernel that kdb will use to resolve kernel symbol definitions. A kernel file must be available. When examining a system dump it is imperative that the kernel file be the same as the kernel that was used to take the system dump. The default for the KernelFile is /unix.

The KernelModule parameters specify the file names of any additional kernel modules which the kdb command uses to resolve symbol definitions not found in the kernel file itself.

Root permissions are required for use of the kdb command on the active system. This is required because the special file /dev/pmem is used. To run the kdb command on the active system, type the following:

```
kdb
```

**Note:** Stack tracing of the current process on a running system does not work.

To invoke the kdb command on a system image file, type:

```
kdb SystemImageFile
```

When kdb starts, it looks for a .kdbinit file in the user’s home directory and in the current working directory. If a .kdbinit file exists in either of these locations, kdb runs all the commands inside the file as if they were entered at the interactive kdb prompt. If a .kdbinit file exists in both of these locations, the file in the home directory will be processed first followed by the file in the current working directory (unless the current directory is the home directory, in which case the file is processed only once).
Flags

- **c CommandFile**  Specifies a different name for the startup script file. If this option is used, then kdb will search for the CommandFile parameter in the home and current directories, instead of the .kdbinit file.

- **-cp**  Causes kdb to print out each command in the startup script files as that command is run. This may be used to aid in the debugging of .kdbinit files (or any other file specified with the -c flag). Each command will be printed with a + (plus) sign in front of it.

- **-h**  Displays a short help message in regard to command line usage and a brief listing of the available command line options.

- **-i HeaderFile**  Makes all of the C structures defined in the HeaderFile parameter available for use with the kdb print subcommand. This option requires a C compiler to be installed on the system. If the HeaderFile variable needs additional .h files to compile, these may have to be specified with separate -i options as well.

- **-k Module**  Instructs kdb to use the Module parameter as an additional kernel module for resolving symbol definitions not found in the kernel itself. Using this option is equivalent to specifying the kernel module with the KernelModule parameter.

- **-l**  Disables the inline pager (that is, the more (^C to quit) ? prompt) in kdb. In this case the set scroll subcommand in kdb has no effect, and the inline pager is always disabled regardless of the scroll setting.

- **-m Image**  Instructs kdb to use the Image parameter as the system image file. Using this option is equivalent to specifying the system image file with the SystemImageFile parameter.

- **-script**  Disables the inline pager (that is, the more (^C to quit) ? prompt) and disables printing of most status information when kdb starts. This option facilitates parsing of the output from the kdb command by scripts and other programs that act as a front end for kdb.

- **-u Kernel**  Instructs kdb to use the Kernel as the kernel file for resolving symbol definitions. Using this option is equivalent to specifying the kernel with the KernelFile parameter.

- **-v**  Displays a list of all Component Dump Tables (CDTs) in the system dump file when kdb starts. CDTs list which memory regions are actually included in the system dump. If kdb is used on a live system, this option is ignored.

- **-w**  Examines a kernel file directly instead of a system image. All kdb subcommands which normally display memory locations from the system image file will instead read data directly from KernelFile. Subcommands which write memory are not available.

Examples

The following examples demonstrate invocation options for the kdb command

1. To invoke the kdb command with the default system image and kernel image files, type:
   ```bash
   kdb
   ```
   The kdb program returns a (0)> prompt and waits for entry of a subcommand.

2. To invoke the kdb command using a dump file named /var/adm/ras/vmcore.0 and the UNIX kernel file named /unix, type:
   ```bash
   kdb /var/adm/ras/vmcore.0 /unix
   ```
   The kdb program returns a (0)> prompt and waits for entry of a subcommand.

Files

/usr/sbin/kdb  Contains the kdb command.
/dev/pmem  Default system image file
/unix  Default kernel file
Appendix B. Kernel extension example files

The section contains information about the following:
- "Loading the kernel extension"
- "Building the demonstration programs"
- "Generating map and list files" on page 444
- "Understanding the compiler list file" on page 444
- "Understanding map files" on page 445
- "Using the comp_link script" on page 446
- "Unloading the demokext kernel extension" on page 451

It also contains various example files.

Loading the kernel extension
To load the demokext kernel extension, complete the following:
1. Run the demonstration program by typing the following:
   
   ./demo
   
   This loads the demokext kernel extension.

   Note: The default prompt at this time is the dollar sign ($)
2. Stop the demonstration program by pressing the Ctrl+Z key sequence.
3. Put the demonstration program in the background by typing the following:
   
   bg
   
4. Activate the KDB kernel debugger using the Ctrl+\ key sequence.
   
   A KDB command prompt should appear. The default KDB prompt is KDB(0)>.

Building the demonstration programs
To build the demonstration program, complete the following:
1. Save the following files in a directory.
   
   - "demo.c Example File" on page 447
   - "demokext.c Example File" on page 448
   - "demo.h Example File" on page 450
   - "demokext.exp example file" on page 451

2. As the root user, run the comp_link script. For more information on the contents of the comp_link
   script, see "comp_link Example File" on page 451.

   This script produces the following:
   
   - An executable file named demo
   - An executable file named demokext
   - A list file named demokext.lst
   - A map file named demokext.map
Generating map and list files

Assembler listing and map files are useful tools for debugging with the KDB kernel debugger. To create the assembler listing during compilation, use the -qlist option. Also use the -qsource option to get the C source listing in the same file. To create the assembler listing with these options, type the following:

```
cc -c -DEBUG -D_KERNEL -DIBMR2 demokext.c -qsource -qlist
```

In order to obtain a map file, use the -bmap:FileName option for the link editor. The following example creates a map file named `demokext.map`.

```
ld -o demokext demokext.o -o-demokext -bimport:/lib/kernex.exp \
  -bmap:demokext.map
```

Understanding the compiler list file

The assembler and source listing is used to correlate any C source line with the corresponding assembler lines. The following is a portion of the list file, created by the `cc` command, for the demonstration kernel extension. This information is included in the compilation listing because the -qsource option for the `cc` command was used. The left column is the line number in the following source code:

```
/* Increment */
scanf(buf, "Before increment: j=%d demokext_j=%d\n", j, demokext_j);
write_log(fpp, buf, &bytes_written);
demokext_j++;
j++;
scanf(buf, "After increment: j=%d demokext_j=%d\n", j, demokext_j);
write_log(fpp, buf, &bytes_written);
break;
```

The assembler listing for the corresponding C code included in the compilation listing because the -qlist option was used with the `cc` command is as follows:

```
63 | case 1: /* Increment */
64 |   scanf(buf, "Before increment: j=%d demokext_j=%d\n", j, demokext_j);
65 |   write_log(fpp, buf, &bytes_written);
66 |   demokext_j++;
67 |   j++;
68 |   scanf(buf, "After increment: j=%d demokext_j=%d\n", j, demokext_j);
69 |   write_log(fpp, buf, &bytes_written);
70 | break;
```

With both the assembler listing and the C source listing, the assembly instructions associated with each C statement can be found. For example, compare the following C source line at line 67 of the demonstration kernel extension.
With the following assembler instructions:

```
  0000E0 1 08830000 1 L4A gr4=demokext_j(gr3,0)
  0000E4 ai 08840001 2 AI gr4=gr4,1
  0000E8 st 08830000 1 ST4A demokext_j(gr3,0)=gr4
```

The offsets of these instructions within the demonstration kernel extension (demokext) are 0000E0, 0000E4, and 0000E8.

**Understanding map files**

The binder map file is a symbol map in address order format. Each symbol listed in the map file has a storage class (CL) and a type (TY) associated with it.

Storage classes correspond to the XMC_TY variables defined in the `syms.h` file. Each storage class belongs to one of the following section types:

- **.text** Contains read-only data (instructions). Addresses listed in this section use the beginning of the .text section as origin. The .text section can contain one of the following storage class (CL) values:
  - DB Debug Table. Identifies a class of sections that has the same characteristics as read only data.
  - GL Glue Code. Identifies a section that has the same characteristics as a program code. This type of section has code to interface with a routine in another module. Part of the interface code requirement is to maintain the table of contents data structure (TOC) addressability across the call.
  - PR Program Code. Identifies the sections that provide executable instructions for the module.
  - R0 Read Only Data. Identifies the sections that contain constants that are not modified while the program is running.
  - TB Reserved for future use.
  - TI Reserved for future use.
  - XO Extended Operations code. Identifies a section of code that is to be treated as a pseudo-machine instruction.

- **.data** Contains read-write initialized data. Addresses listed in this section use the beginning of the .data section as the origin. The .data section can contain one of the following storage class (CL) values:
  - DS Descriptor. Identifies a function descriptor. This information is used to describe function pointers in languages such as C and Fortran.
  - RW Read Write Data. Identifies a section that contains data that is known to require change while the program is running.
  - SV SVC. Identifies a section of code that is to be treated as a supervisory call.
  - T0 TOC Anchor. Used only by the predefined TOC symbol. Identifies the TOC special symbol that is used only by the TOC header.
  - TC TOC Entry. Identifies address data that will reside in the TOC.
  - TD TOC Data Entry. Identifies data that will reside in the TOC.
  - UA Unclassified. Identifies data that contains data of an unknown storage class.

- **.bss** Contains read-write data that is not initialized. Addresses listed in this section use the beginning of the .data section as origin. The .bss section contains one of the following storage class (CL) values:
BS  BSS class. Identifies a section that contains data that is not initialized.
UC  Unnamed Fortran Common. Identifies a section that contains read/write data.

Types correspond to the XTY_TY variables defined in the syms.h file. The type (TY) can be one of the following values:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>External Reference</td>
</tr>
<tr>
<td>LD</td>
<td>Label Definition</td>
</tr>
<tr>
<td>SD</td>
<td>Section Definition</td>
</tr>
<tr>
<td>CM</td>
<td>BSS Common Definition</td>
</tr>
</tbody>
</table>

The following is the map file for the demonstration kernel extension. This file was created because of the -bmap:demokext.map option of the ld command.

```plaintext
ADDRESS MAP FOR demokext

+IE ADDRESS LENGTH AL CL TY Sym#  NAME  SOURCE-FILE(OBJECT) or IMPORT-FILE(SHARED-OBJECT)
--- ------- ---- ---- ---- ----  ------  -----------------------------------------------
 00000000 000000 0  0  0  0  0  _$STATIC  demokext.c(demokext.o)
 00000000 000000 0  0  0  0  0  .demokext  .demokext
 00000000 000000 0  0  0  0  0  .close_log  .close_log
 00000000 000000 0  0  0  0  0  .write_log  .write_log
 00000000 000000 0  0  0  0  0  .open_log  .open_log
 00000000 000000 0  0  0  0  0  .strcpy  .strcpy
 00000000 000000 0  0  0  0  0  .sprintf  .sprintf
 00000000 000000 0  0  0  0  0  .fp_close  .fp_close
 00000000 000000 0  0  0  0  0  .fp_write  .fp_write
 00000000 000000 0  0  0  0  0  .fp_open  .fp_open
 00000000 000000 0  0  0  0  0  _$STATIC  demokext.c(demokext.o)
```

In the above map file, the .data section begins at the statement for line 24:

```
24 00000000 0000F9 3 RW SD S21  <$STATIC>  demokext.c(demokext.o)
```

The TOC (Table Of Contents) starts at the statement for line 27:

```
27 0000010C 000000 2 TO SD S24  <$TOC>
```

Using the comp_link script

The following topics include source code compilation examples and examples of link options used in the comp_link script:

- "demo.c Example File" on page 447
- "demokext.c Example File" on page 448
- "demo.h Example File" on page 450
demo.c Example File

This topic contains an example file that is a source program file that loads, runs, and unloads a demonstration kernel extension.

```c
#include <sys/types.h>
#include <sys/sysconfig.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include "demo.h"

/* Extension loading data */
struct cfg_load cfg_load;
extern int sysconfig();
extern int errno;

#define NAME_SIZE 256
#define LIBPATH_SIZE 256

main(argc,argv)
int argc;
char *argv[];
{
    char path[NAME_SIZE];
    char libpath[LIBPATH_SIZE];
    char buf[BUFLEN];
    struct cfg_kmod cfg_kmod;
    struct extparms extparms = {argc,argv,buf,BUFLEN};
    int option = 1;
    int status = 0;

    /* Load the demo kernel extension. */
    memset(path, 0, sizeof(path));
    memset(libpath, 0, sizeof(libpath));
    strcpy(path, "/demokext");
    cfg_load.path = path;
    cfg_load.libpath = libpath;
    if (sysconfig(SYS_KLOAD, &cfg_load, sizeof(cfg_load)) == CONF_SUCC)
    {
        printf("Kernel extension \demokext was successfully loaded, kmid=\%x\n",
                cfg_load.kmid);
    }
    else
    {
        printf("Encountered errno=\%d loading kernel extension \%s\n",
                errno, cfg_load.path);
        exit(1);
    }

    /* Loop alternately allocating and freeing 16K from memory. */
    option = 1;
    while (option != 0)
    {
        printf("\n\n");
        printf("0. Quit and unload kernel extension
");
        printf("1. Configure kernel extension - increment counter\n");
        printf("2. Configure kernel extension - decrement counter\n");
```
printf("\n");
printf("Enter choice: ");
scanf("%d", &option);
switch (option)
{
    case 0:
        break;
    case 1:
        bzero(buf,BUFLEN);
        strncpy(buf,"sample string");
        cfg_kmod.kmid = cfg_load.kmid;
        cfg_kmod.cmd = 1;
        cfg_kmod.mdiptr = (char *)&extparms;
        cfg_kmod.mdilen = sizeof(extparms);
        if (sysconfig(SYS_CFGKMOD,&cfg_kmod, sizeof(cfg_kmod)) == CONF_SUCCE)
        {
            printf("Kernel extension %s was successfully configured\n", cfg_load.path);
        }
        else
        {
            printf("errno=%d configuring kernel extension %s\n", errno, cfg_load.path);
        }
        break;
    case 2:
        bzero(buf,BUFLEN);
        strncpy(buf,"sample string");
        cfg_kmod.kmid = cfg_load.kmid;
        cfg_kmod.cmd = 2;
        cfg_kmod.mdiptr = (char *)&extparms;
        cfg_kmod.mdilen = sizeof(extparms);
        if (sysconfig(SYS_CFGKMOD,&cfg_kmod, sizeof(cfg_kmod)) == CONF_SUCCE)
        {
            printf("Kernel extension %s was successfully configured\n", cfg_load.path);
        }
        else
        {
            printf("errno=%d configuring kernel extension %s\n", errno, cfg_load.path);
        }
        break;
    default:
        printf("\nUnknown option\n");
        break;
}

/*
 * Unload the demo kernel extension.
 */
if (sysconfig(SYS_KULOAD, &cfg_load, sizeof(cfg_load)) == CONF_SUCCE)
{
    printf("Kernel extension %s was successfully unloaded\n", cfg_load.path);
}
else
{
    printf("errno=%d unloading kernel extension %s\n", errno, cfg_load.path);
}

**demokext.c Example File**

This topic contains an example file that contains the source used to demonstrate the kernel extension.
#include <sys/types.h>
#include <sys/malloc.h>
#include <sys/uio.h>
#include <sys/dump.h>
#include <sys/errno.h>
#include <sys/uprintf.h>
#include <fcntl.h>
#include "demo.h"

/* Log routine prototypes */
ext open_log(char *path, struct file **fpp);
ext write_log(struct file *fpp, char *buf, int *bytes_written);
ext close_log(struct file *fpp);

/* Unexported symbol */
ext demokext_i = 9;
/* Exported symbol */
ext demokext_j = 99;

/ * Kernel extension entry point, called at config. time. *
 * input:
 *  cmd - unused (typically 1=config, 2=unconfig)
 *   uiop - points to the uio structure.
 */
ext demokext(int cmd, struct uio *uiop)
{
    int rc;
    char *bufp;
    struct file *fpp;
    int fstat;
    char buf[100];
    int bytes_written;
    static int j = 0;

    /* Open the log file. */
    strcpy(buf, "./demokext.log");
    fstat = open_log(buf, &fpp);
    if (fstat != 0) return(fstat);

    /* Put a message out to the log file. */
    strcpy(buf, "demokext was called for configuration\n");
    fstat = write_log(fpp, buf, &bytes_written);
    if (fstat != 0) return(fstat);

    /* Increment or decrement j and demokext_j based on
     * the input value for cmd.
     */
    switch (cmd)
    {
    case 1: /* Increment */
        sprintf(buf, "Before increment: j=%d demokext_j=%d\n", j, demokext_j);
        write_log(fpp, buf, &bytes_written);
        demokext_j++;
        j++;
        sprintf(buf, "After increment: j=%d demokext_j=%d\n", j, demokext_j);
write_log(fpp, buf, &bytes_written);
break;

case 2: /* Decrement */
    sprintf(buf, "Before decrement: j=%d demokext_j=%d\n", 
            j, demokext_j);
    write_log(fpp, buf, &bytes_written);
    demokext_j--;
    j--;
    sprintf(buf, "After decrement: j=%d demokext_j=%d\n", 
            j, demokext_j);
    write_log(fpp, buf, &bytes_written);
break;

default: /* Unknown command value */
    sprintf(buf, "Received unknown command of %d\n", cmd);
    write_log(fpp, buf, &bytes_written);
break;
}

/**************************************************************/
*/ Routines for logging debug information: */
/* open_log - Opens a log file */
/* write_log - Output a string to a log file */
/* close_log - Close a log file */
/**************************************************************/
int open_log(char *path, struct file **fpp)
{
    int rc;
    rc = fp_open(path, O_CREAT | O_APPEND | O_WRONLY,
                 S_IRUSR | S_IWUSR, 0, SYSFILES, fpp);
    return(rc);
}

int write_log(struct file *fpp, char *buf, int *bytes_written)
{
    int rc;
    rc = fp_write(fpp, buf, strlen(buf), 0, SYSFILES, bytes_written);
    return(rc);
}

int close_log(struct file *fpp)
{
    int rc;
    rc = fp_close(fpp);
    return(rc);
}

demo.h Example File
This topic contains the code for an include file that is used by the demo.c example file and the demokext.c example file.

#ifndef _demo
#define _demo
/*
 * Parameter structure
*/
*/
struct extparms {
    int argc;
    char **argv;
    char *buf; /* Message buffer */
    size_t len; /* length */
};

#define BUFLEN 4096 /* Test msg buffer length */
#endif /* _demo */

demokext.exp example file
This topic contains the example code that is used as an export file for linking the demokext kernel extension.

#!/unix
* export value from demokext
demokext_j

comp_link Example File
This topic contains an example script that can be used to build the demonstration program and the kernel extension.

#!/bin/ksh
# Script to build the demo executable and the demokext kernel extension.
cc -o demo demo.c
c -c -DDEBUG -D_KERNEL -DIBMR2 demokext.c -qsource -qlist
ld -o demokext demokext.o -edemokext -bimport:/lib/syscalls.exp -bimport:/lib/kernex.exp -lcsys -bexport:demokext.exp -bmap:demokext.map

Unloading the demokext kernel extension

To unload the demokext kernel extension:
1. At the $ prompt, bring the demonstration program to the foreground by typing fg on the command line. At this point, the prompt changes to ./demo.
2. Enter 0 to unload and exit, 1 to increment counters, or 2 to decrement counters. The prompt is not displayed again because it was shown prior to stopping the program and placing it in the background. For the purposes of this example, enter 0 to indicate that the kernel extension is to be unloaded and that the demonstration program is to terminate.
Appendix C. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 003
11400 Burnet Road
Austin, TX 78758-3498
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
(c) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

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

AIX
AIX 5L
IBM
POWER Architecture

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be the trademarks or service marks of others.
## Index

### Special characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>54</td>
</tr>
<tr>
<td>?</td>
<td>8, 43</td>
</tr>
<tr>
<td>*</td>
<td>10</td>
</tr>
<tr>
<td>[</td>
<td>130</td>
</tr>
<tr>
<td>@</td>
<td>8, 10</td>
</tr>
</tbody>
</table>

### A

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>address translation</td>
<td></td>
</tr>
<tr>
<td>subcommands</td>
<td></td>
</tr>
<tr>
<td>ibat</td>
<td>247</td>
</tr>
<tr>
<td>mdbat</td>
<td>248</td>
</tr>
<tr>
<td>mibat</td>
<td>249</td>
</tr>
<tr>
<td>msib</td>
<td>245</td>
</tr>
<tr>
<td>slb</td>
<td>243</td>
</tr>
<tr>
<td>tr</td>
<td>242, 246</td>
</tr>
<tr>
<td>tv</td>
<td>242</td>
</tr>
<tr>
<td>ames</td>
<td>167</td>
</tr>
<tr>
<td>apt</td>
<td>169</td>
</tr>
<tr>
<td>assembler listing</td>
<td>444</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>116</td>
</tr>
<tr>
<td>B</td>
<td>124, 125</td>
</tr>
<tr>
<td>basic display</td>
<td></td>
</tr>
<tr>
<td>subcommands</td>
<td>75</td>
</tr>
<tr>
<td>f</td>
<td>76</td>
</tr>
<tr>
<td>pr</td>
<td>83</td>
</tr>
<tr>
<td>print</td>
<td>83</td>
</tr>
<tr>
<td>stack</td>
<td>76</td>
</tr>
<tr>
<td>stat</td>
<td>81</td>
</tr>
<tr>
<td>status</td>
<td>80</td>
</tr>
<tr>
<td>symptom</td>
<td>87</td>
</tr>
<tr>
<td>where</td>
<td>76</td>
</tr>
<tr>
<td>bdev</td>
<td>436</td>
</tr>
<tr>
<td>bmb</td>
<td>361</td>
</tr>
<tr>
<td>bmbblk</td>
<td>361</td>
</tr>
<tr>
<td>bmblock</td>
<td>361</td>
</tr>
<tr>
<td>bosboot</td>
<td>2, 3</td>
</tr>
<tr>
<td>bqueue</td>
<td>439</td>
</tr>
<tr>
<td>branch target</td>
<td></td>
</tr>
<tr>
<td>subcommands</td>
<td>137</td>
</tr>
<tr>
<td>btac</td>
<td>137, 138</td>
</tr>
<tr>
<td>ctac</td>
<td>138</td>
</tr>
<tr>
<td>ibtac</td>
<td>138</td>
</tr>
<tr>
<td>lcibtac</td>
<td>138</td>
</tr>
<tr>
<td>breakpoint</td>
<td></td>
</tr>
<tr>
<td>setting</td>
<td>21</td>
</tr>
<tr>
<td>breakpoints</td>
<td></td>
</tr>
<tr>
<td>subcommands</td>
<td>115</td>
</tr>
<tr>
<td>b</td>
<td>116</td>
</tr>
<tr>
<td>brk</td>
<td>116</td>
</tr>
<tr>
<td>c</td>
<td>120</td>
</tr>
<tr>
<td>ca</td>
<td>120</td>
</tr>
<tr>
<td>cl</td>
<td>120</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>120</td>
</tr>
<tr>
<td>ca</td>
<td>120</td>
</tr>
<tr>
<td>calculator</td>
<td></td>
</tr>
<tr>
<td>subcommands</td>
<td>69</td>
</tr>
<tr>
<td>cal</td>
<td>70</td>
</tr>
<tr>
<td>conv</td>
<td>71</td>
</tr>
<tr>
<td>dcal</td>
<td>70</td>
</tr>
<tr>
<td>hcal</td>
<td>70</td>
</tr>
<tr>
<td>cat</td>
<td>131</td>
</tr>
<tr>
<td>cc</td>
<td>444</td>
</tr>
<tr>
<td>cdt</td>
<td>390</td>
</tr>
<tr>
<td>change context</td>
<td></td>
</tr>
<tr>
<td>subcommands</td>
<td>59</td>
</tr>
<tr>
<td>context</td>
<td>64</td>
</tr>
<tr>
<td>cpu</td>
<td>63</td>
</tr>
<tr>
<td>ctx</td>
<td>64</td>
</tr>
<tr>
<td>sw</td>
<td>60</td>
</tr>
<tr>
<td>switch</td>
<td>60</td>
</tr>
<tr>
<td>check</td>
<td>393</td>
</tr>
<tr>
<td>cl</td>
<td>120</td>
</tr>
<tr>
<td>cla</td>
<td>434</td>
</tr>
<tr>
<td>class</td>
<td>434</td>
</tr>
<tr>
<td>clk</td>
<td>398</td>
</tr>
<tr>
<td>command line</td>
<td></td>
</tr>
<tr>
<td>editing</td>
<td>11</td>
</tr>
<tr>
<td>commands</td>
<td></td>
</tr>
<tr>
<td>entering</td>
<td></td>
</tr>
<tr>
<td>KDB kernel debugger</td>
<td>4</td>
</tr>
<tr>
<td>compiler</td>
<td></td>
</tr>
<tr>
<td>list file</td>
<td>444</td>
</tr>
<tr>
<td>context</td>
<td>64</td>
</tr>
<tr>
<td>context information, display</td>
<td>261</td>
</tr>
<tr>
<td>subcommands</td>
<td>261</td>
</tr>
<tr>
<td>cr</td>
<td>281</td>
</tr>
<tr>
<td>cred</td>
<td>285</td>
</tr>
<tr>
<td>crid</td>
<td>281</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1997, 2006
context information, display (continued)  
subcommands (continued)  
lastbackt 268
lockq 280
lq 280
mst 266
pnda 262
ppda 264
ppid 270
pr_pid 270
pr_tid 271
ptid 271
rq 276
rqa 278
rqi 278
runq 276
svmon 282
th_pid 275
th_tid 273
tpid 275
ttid 273
conv 71
convertor
subcommands 69
cal 70
conv 71
dcal 70
hcal 70
cpu 7, 63
CPU
subcommands 73
start 74
stop 74
cr 281
cred 285
crid 281
CSA 63
c t 131
c t ac 138
c tctrl 395
c tx 64
cu 5
cupboard 372

d dvw 98, 99
d debug 52
d debug, I/O
subcommands 145
buserr 149
businfo 152
dpcib 146
dpchi 146
dpciw 146
mpcib 147
mpchi 147
mpciw 147
d debugger prompt 7
decode 94
demonstration program
building 443
dereference 10
dev 161
devno 340
devnode 340
devsw 24, 161
di 94
diob 98, 99
diod 98, 99
diow 98, 99
dis 92
d display context information
subcommands

cr 281
cred 285
crid 281
lastbackt 268
lockq 280
lq 280
mst 266
pnda 262
ppda 264
ppid 270
pr_pid 270
pr_tid 271
ptid 271
rq 276
rqa 278
rqi 278
runq 276
svmon 282
th_pid 275
th_tid 273
tpid 275
ttid 273
d display general file system information
subcommands

dbuf 317
d buffer 317
devno 340
devnode 340
dnlc 312
d fb 320
dbuffer 320
dfifon 342

D

d 90
d DABR
subcommands 133
dbat 246
dbgopt 47
dc 92
dcal 70
dd 90
ddoor 98, 99
ddpd 98, 99
ddph 98, 99
ddpw 98, 99
ddvb 98, 99
ddvd 98, 99
ddvh 98, 99
display general file system information  (continued)
subcommands  (continued)
  fifonode  342
  file  323
  fino  329
  gfs  322
  gno  321
  gnode  321
  hb  319
  hbuffer  319
  hino  328
  hinode  328
  hno  344
  hnode  344
  hvncache  333
  hvnc  333
  icache  329
  ino  325
  inode  325
  jfsnode  345
  jno  345
  kfs  347
  kfset  347
  knv  316
  mount  336
  ncache  312
  specno  338
  specnode  338
  vfs  336
  vno  335
  vnode  335

display j2 file system information
subcommands
  bmb  361
  bmbblk  361
  bmblock  361
  dt  354
  dtree  354
  i2  351
  inode2  351
  j2  350
  j2log  366
  j2logbuf  364
  j2logx  365
  j2no  362
  jfs2  350
  jfs2node  362
  pbuf  357
  pgobj  356
  pile  367
  slab  369
  tree  353
  txblk  358
  txblki  359
  txblock  358
  txblocki  359
  txlock  360
  xt  355
  xtree  355

display kernel data structures
subcommands
  dev  161
  devsw  161
  drvars  158
  intr  162
  ipl  159
  iplcb  159
  var  156
  zproc  237

display memory allocation information
subcommands
  bucket  306
  hp  297
  kmbucket  306
  kmstats  308
  xm  303
  xmalloc  303

display NFS information
subcommands
  cupboard  372
  sockcup  375
  sockpint  374
  svcxprt  376

display storage subsystem information
subcommands
  lv  292
  pbuf  288
  pv  291
  scd  293
  scdisk  293
  volgrp  289

display VMM
subcommands
  ames  167
  apt  169
  drl  238
  drlist  238
  framset  171
  free  172
  frs  171
  ipc  174
  ksp  211
  lka  180
  lk  181
  lw  182
  lockanch  180
  lockhash  181
  lockword  182
  lrustate  239
  mem  184
  mempool  184
  pdt  186
  pfhtable  188
  pft  190
  pta  198
  pvlist  196
  pvt  196
  rmap  201
  rtipc  176
  rtipcd  178
display VMM (continued)
subcommands (continued)
  rsid 203
  scb 204
  segst64 207
  sidd 235
  sidm 236
  sr64 208
  ste 213
  swhat 194
  tblk 180
  vl 225
  vmdaddr 217
  vmbufst 215
  vmdmap 218
  vmint 220
  vmker 223
  vmlock 225
  vmlocks 225
  vmlog 227
  vmpool 228
  vmstat 230
  vmthrpgio 232
  vmwait 233
  vrlid 234
  vsidd 235
  vsidm 236

display_stack_frames 18

display, basic
  subcommands 75
    f 76
    pr 83
    print 83
    stack 76
    stat 81
    status 80
    symptom 87
    where 76

displaying system images
  for examining a dump
  using kdb command 441

dump (continued)
  subcommands (continued)
    trace 384
    trcstart 386
    trcstop 387

dump, examining
  displaying system images
    using kdb command 441

dw 4, 90

dw demokext_j 13

E

e 56

editing
  command line 11
    emacs 11
    gmacs 11
    vi 12

    emacs 11

    enabling KDB kernel debugger
      AIX 4.3.3 3
      AIX 5.1 2

    end user
      subcommands 41
        debug 52
        h 41, 42
        hi 51
        his 51
        hist 51
        varlist 49
        varrm 50
        varset 48

error log
  subcommands 383
    cdt 390
    errpt 392
    trace 384
    trcstart 386
    trcstop 387

erpt 392

example files
  kernel extension 443
  exp 259
  expressions 10
  ext 104
  exp 104

extract
  subcommands 101

F

  f 76, 77
  fb 320
  fbuffer 320
  fifono 342
  fifonode 342
  file 323

  find 102, 103
  findp 102, 103
  fino 329
G

general file system information, display subcommands
buf 317
buffer 317
devote 340
devnode 340
dnlc 312
fb 320
fbuffer 320
fifono 342
fifonode 342
file 323
fino 329
gfs 322
gno 321
gnode 321
hb 319
hbuffer 319
hino 328
hinode 328
hno 344
hnnode 344
hvcache 333
hvnc 333
icache 329
ino 325
inode 325
jfsnode 345
jno 345
kfs 347
kfset 347
kvn 316
mount 336
ncache 312
specno 338
specnode 338
vfs 336
vno 335
vnode 335

general information, display file system subcommands 311
generating
list files 444
map files 444
gfs 322
global data
modifying 13
map file 15
symbol name 13
TOC and map file 14
viewing 13
map file 15
symbol name 13

global data (continued)
viewing (continued)
TOC and map file 14
gmacs 11
gno 321
gnode 321
gt 122

H

h 42, 43
halt 58
hb 319
hbuffer 319
hcal 15, 70
hbnlc 314
heap 298
help 43
online 7
hi 51
hino 328
hinode 328
his 51
hist 51
HMC 4
hno 344
hnnode 344
hvncache 333
hvnc 333

I

i2 351
ibat 247
icache 329
ifnet 402
information, display context
subcommands 261
cr 281
cred 285
crid 281
lastback 268
lockq 280
lq 280
mst 266
pnda 262
ppda 264
ppid 270
pr_pid 270
pr_tid 271
prid 271
rq 276
rqa 278
rqi 278
runq 276
svmon 282
th_pid 275
th_tid 273
tpid 275
ttid 273
information, display general file system
subcommands 311
buf 317
buffbuffer 320
buffer 317
devno 340
devnode 340
dnlc 312
db 320
fifono 342
fifonode 342=file 323
fino 329
gs 322
gno 321
gnode 321
fb 320
hbuffer 319
hino 328
hinode 328
hno 344
hnode 344
hc 329
icache 329
ino 325
inode 325
jfsnode 345
jno 345
kfs 347
kset 347
kv 316
mount 336
ncache 312
specno 338
specnode 338
vcs 336
vno 335
vnode 335

information, display j2 file system
subcommands 349
bmb 361
bmbblk 361
bmblock 361
dt 354
dtree 354
i2 351
inode2 351
j2 350
j2log 366
j2logbuf 364
j2logx 365
j2no 362
jfs2 350
jfs2node 362
pgbuf 357
pgobj 356
pile 367
slab 369
tree 353
txblk 358

information, display j2 file system (continued)
subcommands (continued)
txblk 359
txblock 358
txblocki 359
txlp 360
txlock 360
xt 355
xtree 355

information, display memory allocation
subcommands 297

information, display NFS
subcommands 371
cupboard 372
sockcup 375
sockpint 374
svcxp 376

information, display storage subsystem
subcommands 287
lvol 292
pbuf 288
pvol 391
scd 293
scdisk 293
volgrp 289

information, memory allocation subsystem
subcommands
bucket 306
hp 297
kmbucket 306
kmstats 306
xm 303
xmalloc 303
ino 325
inode 325
inode2 351
intr 162

introduction
kdb command 5
KDB kernel debugger 1

invoking
kdb command 6
ipc 174
ipl 159
iplob 159

J
j2 350
j2 file system information, display
subcommands 349
bmb 361
bmbblk 361
bmblock 361
dt 354
dtree 354
i2 351
inode2 351
j2 350
j2log 366
j2logbuf 364
<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>j2logx</td>
<td>365</td>
</tr>
<tr>
<td>j2no</td>
<td>362</td>
</tr>
<tr>
<td>jfs2</td>
<td>350</td>
</tr>
<tr>
<td>jfs2node</td>
<td>362</td>
</tr>
<tr>
<td>pgbuf</td>
<td>357</td>
</tr>
<tr>
<td>pgobj</td>
<td>356</td>
</tr>
<tr>
<td>pile</td>
<td>367</td>
</tr>
<tr>
<td>slab</td>
<td>369</td>
</tr>
<tr>
<td>tree</td>
<td>353</td>
</tr>
<tr>
<td>txblk</td>
<td>358</td>
</tr>
<tr>
<td>txblki</td>
<td>359</td>
</tr>
<tr>
<td>txblock</td>
<td>358</td>
</tr>
<tr>
<td>txblocki</td>
<td>359</td>
</tr>
<tr>
<td>txick</td>
<td>360</td>
</tr>
<tr>
<td>txlock</td>
<td>360</td>
</tr>
<tr>
<td>xt</td>
<td>355</td>
</tr>
<tr>
<td>xtree</td>
<td>355</td>
</tr>
<tr>
<td>j2log</td>
<td>366</td>
</tr>
<tr>
<td>j2logbuf</td>
<td>364</td>
</tr>
<tr>
<td>j2logx</td>
<td>365</td>
</tr>
<tr>
<td>j2no</td>
<td>362</td>
</tr>
<tr>
<td>jfs2</td>
<td>350</td>
</tr>
<tr>
<td>jfs2node</td>
<td>362</td>
</tr>
<tr>
<td>jfsnode</td>
<td>345</td>
</tr>
<tr>
<td>jno</td>
<td>345</td>
</tr>
</tbody>
</table>

**K**

- kdb command 441
  - introduction 1
- KDB Debugger trace
  - subcommands 127
- KDB kernel debugger
  - example files 447, 448, 450, 451
  - introduction 1
  - subcommands 29
- kernel data structures, display
  - subcommands 155
    - drvvars 158
    - var 156
- kernel extension example files 443
- key
  - sequence 11
- kfs 347
- kfset 347
- kmbucket 306
- kmid 23
- kmstats 308
- ksp 211
- kvn 316

**L**

- lastbackt 268
- lb 118
- lbrk 118
- lbtac 138
- lc 120
- lcbtac 138

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>j2 log</td>
<td>366</td>
</tr>
<tr>
<td>j2 log buf</td>
<td>364</td>
</tr>
<tr>
<td>j2 log x</td>
<td>365</td>
</tr>
<tr>
<td>j2 no</td>
<td>362</td>
</tr>
<tr>
<td>jfs 2</td>
<td>350</td>
</tr>
<tr>
<td>jfs 2 node</td>
<td>362</td>
</tr>
<tr>
<td>pg buf</td>
<td>357</td>
</tr>
<tr>
<td>pg obj</td>
<td>356</td>
</tr>
<tr>
<td>pile</td>
<td>367</td>
</tr>
<tr>
<td>slab</td>
<td>369</td>
</tr>
<tr>
<td>tree</td>
<td>353</td>
</tr>
<tr>
<td>tx blk</td>
<td>358</td>
</tr>
<tr>
<td>tx bl ki</td>
<td>359</td>
</tr>
<tr>
<td>tx block</td>
<td>358</td>
</tr>
<tr>
<td>tx block i</td>
<td>359</td>
</tr>
<tr>
<td>txick</td>
<td>360</td>
</tr>
<tr>
<td>tx lock</td>
<td>360</td>
</tr>
<tr>
<td>xt</td>
<td>355</td>
</tr>
<tr>
<td>xtree</td>
<td>355</td>
</tr>
<tr>
<td>lock</td>
<td>397</td>
</tr>
<tr>
<td>lock anch</td>
<td>180</td>
</tr>
<tr>
<td>lock hash</td>
<td>181</td>
</tr>
<tr>
<td>lock q</td>
<td>280</td>
</tr>
<tr>
<td>lock word</td>
<td>280</td>
</tr>
<tr>
<td>lr u</td>
<td>239</td>
</tr>
<tr>
<td>lr u state</td>
<td>239</td>
</tr>
<tr>
<td>lv o l</td>
<td>292</td>
</tr>
</tbody>
</table>

**M**

- m 108
- map file 445
- map files
  - generating 444
- mathematical operators 10
- mbuf 412
- md 108
- m db at 248
- m db p b 111, 112
- m dp d 111, 112
- m dp h 111, 112
- m dv b 111, 112
- m dv d 111, 112

Index 461
memory register (continued)
        subcommands (continued)
          mw 108
memory, extract
        subcommands
          ext 104
          extp 104
memory, search
        subcommands
          find 102
          findp 102
memp 184
mempool 184
mibat 249
miob 111, 112
mioid 111, 112
miow 111, 112
mode
          edit 11
modifying
          global data 13
            map file 15
            symbol name 13
            TOC and map file 14
more 7
mount 336
mp 108
mpcib 147
mpch 147
mpciv 147
mpd 108
mpw 108
mr 113, 388
mslb 245
mst 266
mtrace 388
multiprocessor
        system
          number 12
          mw 108
n
  n 124, 125
namelist
        subcommands 141
ncache 312
ndd 417
net
        subcommands
          ifnet 402, 417
          mbuf 412
          netm 413
          netstat 420
          nsdbg 419
          route 422
          rxnode 427
          si 414
          sock 408
          sockinfo 414
net (continued)
subcommands (continued)
tcb 404
tcpb 410
ubd 406
netm 413
netstat 420, 421
network
subcommands 401
NFS information, display
subcommands 371
cupboard 372
sockcup 375
sockpint 374
svcxpr 376
nm 142
ns 143
nsdbg 419

O
offsets 445
online help 7

P
pause 11
pbuf 288
PCI
subcommands
buser 149
businfo 152
dpcib 146
dpcih 146
dpciw 146
mpcib 147
mpcih 147
mpciw 147
pdt 186
pfhdata 188
pft 190
pgbuf 357
pgobj 356
pile 367
pnda 262
ppda 264
PPDA 63
ppid 270
pr 83
pr_pid 270
pr_tid 271
print 83
prompt
debugger 7
pta 198
pte 199
ptid 271
pvlist 196
pvol 291
pvt 196

Q
q 56

R
r 122
reboot 57
recursive
dereference 10
registers 8
repeat count 11
resume 11
return 122
rmap 201
rmt 252, 253
route 422
rq 276
rqa 278
rqi 278
rtentry 425
rtipc 176
rtipcd 178
rule 440
rules 440
runcpu 67
runq 276
rvisd 203
rxnode 427

S
s 124, 125
S 124, 125
scb 204
scd 293
scdisk 293
screen size 7
scroll 7
search
subcommands 101
segst64 207
sequence
key 11
set 11, 44, 45
setting
breakpoint 21
setup 44, 45
si 414
sidd 235
sidm 236
slab 369
slb 243
slk 398
sock 408
sockcup 375
sockinfo 414
sockpint 374
specno 338
specnode 338
sr64 208
subcommand category (continued)

KDB Debugger trace
   [ 130
   bt 128
   cat 131
   ct 131
   test 130
KDB Debugger trace points 127
   leave 55
   loader 251
   lock 397
memory modification 107
memory register 89
namelist 141
network 401
PCI 145
search 101
steps 115
symbols 141
system dump 383
system error log 383
system trace 383
time 377
trace points 127
watch 133
WLM 433
svcxprt 376
svmon 282
sw 60
swhat 194
switch 60
symbols
   subcommands 141
      nm 142
      ns 143
      ts 142
   symptom 87
system dump
   subcommands 383
      cdt 390
      erpt 392
      trace 384
      trcstart 386
      trcstop 387
system error log
   subcommands 383
      cdt 390
      erpt 392
      trace 384
      trcstart 386
      trcstop 387
system hang 5
system images, displaying
   for examining a dump
      using kdb command 441
system trace
   subcommands 383
      cdt 390
      erpt 392
      trace 384
      trcstart 386
system trace (continued)
subcommands (continued)
trcstop 387

T
tblk 180
tcb 404
tcpb 410
tcpdbg 429
terminal 4
test 130
th_pid 275
th_tid 273
time 378
subcommands 377
time 378
tpid 275
tr 242
trace 27, 384, 385
subcommands
cdt 390
erpt 392
trace 384
trcstart 386
trcstop 387
trace points
subcommands 127
translation, address
subcommands 241
trb 380
trcdead 27
trcstart 386
trcstop 387
tree 353
ts 142
ttid 273
tv 242
txblk 358
txblli 359
txblock 358
txblocki 359
txlock 360

U
udb 406
unload
demokext 451
user-defined
variables 10

V
values
register 8
var 156
variables
user-defined 10
varlist 10, 49
VMM, display (continued)
  subcommands (continued)
    vmint 220
    vmker 223
    vmlock 225
    vmlocks 225
    vmlog 227
    vmpool 228
    vmstat 230
    vmthrgpio 232
    vmwait 233
    vrid 234
    vsidd 235
    vsidm 236
  vmpool 228
  vmstat 230
  vmthrgpio 232
  vmwait 233
  vnc 331
  vno 335
  vnode 335
  volgrp 289
  vrid 234
  vsidd 235
  vsidm 236

W
  watch
    subcommands 133
  wf 144
  where 76, 77
  which 144
  WLM
    subcommands 433
      bdev 436
      bqueue 439
      cla 434
      class 434
      rule 440
      rules 440
      wlm_bdev 436
      wlm_bq 439
      wlm_bdev 436
      wlm_bq 439
      wr 134, 135

X
  xm 303
  xmalloc 303
  xt 355
  xtree 355

Z
  z 167, 292, 386, 393
  zproc 237
Readers’ Comments — We’d Like to Hear from You

AIX 5L Version 5.3
KDB kernel debugger and kdb command

Publication No. SC23-4924-03

Overall, how satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th>Overall satisfaction</th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How satisfied are you that the information in this book is:

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to understand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Well organized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Name

Address

Company or Organization

Phone No.