OpenVMS Alpha System Analysis Tools Manual

Order Number: AA-REZTB-TE

April 2001

This manual explains how to use various Alpha system analysis tools to investigate system failures and examine a running Compaq OpenVMS system.

Revision/Update Information: This manual supersedes the OpenVMS Alpha System Analysis Tools Manual Version 7.2

Software Version: OpenVMS Alpha Version 7.3

Compaq Computer Corporation
Houston, Texas
The Compaq OpenVMS documentation set is available on CD-ROM.
# Contents

<table>
<thead>
<tr>
<th>Preface</th>
<th>xi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Overview of System Analysis Tools</td>
<td>1–1</td>
</tr>
<tr>
<td>1.1 System Dump Analyzer (SDA)</td>
<td>1–1</td>
</tr>
<tr>
<td>1.2 System Code Debugger (SCD)</td>
<td>1–1</td>
</tr>
<tr>
<td>1.3 System Dump Debugger (SDD)</td>
<td>1–2</td>
</tr>
<tr>
<td>1.4 Watchpoint Utility</td>
<td>1–2</td>
</tr>
<tr>
<td>1.5 Delta/XDelta Debugger</td>
<td>1–2</td>
</tr>
<tr>
<td>1.6 Dump Off System Disk (DOSD)</td>
<td>1–3</td>
</tr>
</tbody>
</table>

## Part I OpenVMS Alpha System Dump Analyzer (SDA)

<table>
<thead>
<tr>
<th>2 SDA Description</th>
<th>2–1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Capabilities of SDA</td>
<td>2–1</td>
</tr>
<tr>
<td>2.2 System Management and SDA</td>
<td>2–3</td>
</tr>
<tr>
<td>2.2.1 Writing System Dumps</td>
<td>2–3</td>
</tr>
<tr>
<td>2.2.1.1 Dump File Style</td>
<td>2–3</td>
</tr>
<tr>
<td>2.2.1.2 Comparison of Full and Selective Dumps</td>
<td>2–4</td>
</tr>
<tr>
<td>2.2.1.3 Controlling the Size of Page Files and Dump Files</td>
<td>2–5</td>
</tr>
<tr>
<td>2.2.1.4 Writing to the System Dump File</td>
<td>2–5</td>
</tr>
<tr>
<td>2.2.1.5 Writing to the Dump File off the System Disk</td>
<td>2–6</td>
</tr>
<tr>
<td>2.2.1.6 Writing to the System Page File</td>
<td>2–7</td>
</tr>
<tr>
<td>2.2.2 Saving System Dumps</td>
<td>2–8</td>
</tr>
<tr>
<td>2.2.3 Invoking SDA When Rebooting the System</td>
<td>2–8</td>
</tr>
<tr>
<td>2.3 Analyzing a System Dump</td>
<td>2–9</td>
</tr>
<tr>
<td>2.3.1 Requirements</td>
<td>2–10</td>
</tr>
<tr>
<td>2.3.2 Invoking SDA</td>
<td>2–10</td>
</tr>
<tr>
<td>2.3.3 Mapping the Contents of the Dump File</td>
<td>2–10</td>
</tr>
<tr>
<td>2.3.4 Building the SDA Symbol Table</td>
<td>2–11</td>
</tr>
<tr>
<td>2.3.5 Executing the SDA Initialization File (SDA$INIT)</td>
<td>2–11</td>
</tr>
<tr>
<td>2.4 Analyzing a Running System</td>
<td>2–12</td>
</tr>
<tr>
<td>2.5 SDA Context</td>
<td>2–12</td>
</tr>
<tr>
<td>2.6 SDA Command Format</td>
<td>2–14</td>
</tr>
<tr>
<td>2.6.1 General Command Format</td>
<td>2–14</td>
</tr>
<tr>
<td>2.6.2 Expressions</td>
<td>2–14</td>
</tr>
<tr>
<td>2.6.2.1 Radix Operators</td>
<td>2–15</td>
</tr>
<tr>
<td>2.6.2.2 Arithmetic and Logical Operators</td>
<td>2–15</td>
</tr>
<tr>
<td>2.6.2.3 Precedence Operators</td>
<td>2–16</td>
</tr>
<tr>
<td>2.6.2.4 Symbols</td>
<td>2–16</td>
</tr>
<tr>
<td>2.7 Investigating System Failures</td>
<td>2–21</td>
</tr>
<tr>
<td>2.7.1 General Procedure for Analyzing System Failures</td>
<td>2–22</td>
</tr>
</tbody>
</table>
2.7.2 Fatal Bugcheck Conditions .................................................. 2–22
2.7.2.1 Mechanism Array ......................................................... 2–23
2.7.2.2 Signal Array .............................................................. 2–25
2.7.2.3 64-Bit Signal Array ..................................................... 2–27
2.7.2.4 Exception Stack Frame .................................................. 2–28
2.7.2.5 SSRVEXCEPT Example ................................................ 2–28
2.7.2.6 Illegal Page Faults ...................................................... 2–33
2.8 Inducing a System Failure .................................................... 2–33
2.8.1 Meeting Crash Dump Requirements .................................... 2–34
2.8.2 Procedure for Causing a System Failure ............................... 2–34

3 ANALYZE Usage Summary and Qualifiers
3.1 ANALYZE Usage Summary .................................................. 3–1
3.2 ANALYZE Qualifiers .......................................................... 3–2
   /CRASH_DUMP ......................................................... 3–3
   /OVERRIDE ........................................................... 3–4
   /RELEASE ............................................................ 3–5
   /SYMBOL ............................................................... 3–6
   /SYSTEM ............................................................... 3–7

4 SDA Commands
   @(Execute Command) ......................................................... 4–3
   ATTACH ................................................................. 4–4
   COPY ................................................................. 4–5
   DEFINE ................................................................. 4–7
   DEFINE/KEY ........................................................... 4–9
   DUMP ................................................................. 4–12
   EVALUATE ............................................................... 4–15
   EXAMINE ............................................................... 4–18
   EXIT ................................................................. 4–22
   FORMAT ............................................................... 4–23
   HELP ................................................................. 4–26
   MAP ................................................................. 4–28
   MODIFY DUMP ........................................................... 4–31
   READ ................................................................. 4–33
   REPEAT ................................................................. 4–41
   SEARCH ............................................................... 4–43
   SET CPU ............................................................... 4–45
   SET ERASE_SCREEN ................................................... 4–47
   SET FETCH ............................................................ 4–48
   SET LOG ............................................................... 4–50
   SET OUTPUT .......................................................... 4–51
   SET PROCESS .......................................................... 4–53
   SET RMS ............................................................... 4–56
   SET SIGN_EXTEND ..................................................... 4–59
   SET SYMBOLIZE ....................................................... 4–60
   SHOW ADDRESS ......................................................... 4–61
SHOW BUGCHECK ................................................. 4–63
SHOW CALL_FRAME ............................................. 4–64
SHOW CLUSTER .................................................. 4–66
SHOW CONNECTIONS ........................................... 4–72
SHOW CPU .................................................... 4–74
SHOW CRASH ................................................... 4–77
SHOW DEVICE .................................................. 4–81
SHOW DUMP .................................................... 4–85
SHOW EXECUTIVE .............................................. 4–88
SHOW GALAXY .................................................. 4–92
SHOW GCT ..................................................... 4–93
SHOW GLOBAL_SECTION_TABLE, SHOW GST ................. 4–96
SHOW GLOCK .................................................. 4–98
SHOW GMDB ................................................... 4–101
SHOW GSD ...................................................... 4–103
SHOW HEADER .................................................. 4–105
SHOW LAN ..................................................... 4–106
SHOW LOCKS ................................................... 4–116
SHOW MACHINE_CHECK ....................................... 4–121
SHOW MEMORY .................................................. 4–123
SHOW PAGE_TABLE ............................................ 4–125
SHOW PARAMETER ............................................. 4–131
SHOW PFN_DATA ................................................. 4–134
SHOW POOL .................................................... 4–139
SHOW PORTS .................................................... 4–146
SHOW PROCESS .................................................. 4–150
SHOW RAD ...................................................... 4–173
SHOW RESOURCES .............................................. 4–175
SHOW RMD ...................................................... 4–180
SHOW RMS ...................................................... 4–182
SHOW RSPID .................................................... 4–183
SHOW SHM_CPP .................................................. 4–185
SHOW SHM_REG .................................................. 4–188
SHOW SPINLOCKS ................................................. 4–190
SHOW STACK .................................................... 4–196
SHOW SUMMARY .................................................. 4–200
SHOW SYMBOL .................................................. 4–203
SHOW TQE ....................................................... 4–205
SHOW WORKING_SET_LIST, SHOW WSL ....................... 4–207
SPAWN .......................................................... 4–208
UNDEFINE ....................................................... 4–210
VALIDATE PFN_LIST ............................................ 4–211
VALIDATE QUEUE ................................................. 4–213
VALIDATE SHM_CPP ............................................. 4–215
5 SDA CLUE Extension Commands

5.1 Overview of SDA CLUE Extensions ................................................................. 5–1
5.2 Displaying Data Using SDA CLUE Commands ........................................... 5–2
5.3 Using SDA CLUE with DOSD ............................................................. 5–2
5.4 Listing of SDA CLUE Extension Commands ........................................... 5–3
   CLUE CALL_FRAME ........................................................................... 5–4
   CLUE CLEANUP ............................................................................. 5–7
   CLUE CONFIG .............................................................................. 5–8
   CLUE CRASH ............................................................................... 5–9
   CLUE ERRLOG ............................................................................. 5–12
   CLUE FRU ................................................................................. 5–13
   CLUE HISTORY ........................................................................... 5–14
   CLUE MCHK ............................................................................. 5–16
   CLUE MEMORY ........................................................................... 5–17
   CLUE PROCESS .......................................................................... 5–25
   CLUE REGISTER .......................................................................... 5–27
   CLUE SG .................................................................................. 5–29
   CLUE STACK ............................................................................. 5–30
   CLUE SYSTEM .......................................................................... 5–33
   CLUE VCC ................................................................................ 5–34
   CLUE XQP ................................................................................ 5–37

6 SDA Spinlock Tracing Utility

6.1 Overview of the SDA Spinlock Tracing Utility ............................................. 6–1
6.2 How to Use the SDA Spinlock Tracing Utility ............................................ 6–2
6.3 Example Command Procedure for Collection of Spinlock Statistics .......... 6–3
6.4 Listing of SDA Spinlock Tracing Commands ............................................. 6–3
   SPL LOAD ................................................................................ 6–4
   SPL SHOW COLLECT ................................................................ 6–5
   SPL SHOW TRACE .................................................................... 6–6
   SPL START COLLECT ............................................................. 6–11
   SPL START TRACE ................................................................. 6–12
   SPL STOP COLLECT ............................................................... 6–14
   SPL STOP TRACE ................................................................. 6–15
   SPL UNLOAD ........................................................................... 6–16

7 SDA Extension Routines

7.1 Introduction ......................................................................................... 7–1
7.2 General Description ........................................................................... 7–1
7.3 Detailed Description ......................................................................... 7–2
7.3.1 Compiling and Linking an SDA Extension ..................................... 7–2
7.3.2 Invoking an SDA Extension ........................................................... 7–3
7.3.3 Contents of an SDA Extension ...................................................... 7–3
7.4 Debugging an Extension ................................................................. 7–5
7.5 Callable Routines Overview ............................................................. 7–6
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>Callable Routines Specifics</td>
<td>7–8</td>
</tr>
<tr>
<td>SDA$ADD_SYMBOL</td>
<td>7–9</td>
<td></td>
</tr>
<tr>
<td>SDA$ALLOCATE</td>
<td>7–10</td>
<td></td>
</tr>
<tr>
<td>SDA$DBG_IMAGE_INFO</td>
<td>7–11</td>
<td></td>
</tr>
<tr>
<td>SDA$DEALLOCATE</td>
<td>7–12</td>
<td></td>
</tr>
<tr>
<td>SDA$DISPLAY_HELP</td>
<td>7–13</td>
<td></td>
</tr>
<tr>
<td>SDA$ENSURE</td>
<td>7–15</td>
<td></td>
</tr>
<tr>
<td>SDA$FORMAT</td>
<td>7–16</td>
<td></td>
</tr>
<tr>
<td>SDA$FORMAT_HEADING</td>
<td>7–18</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_ADDRESS</td>
<td>7–19</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_BLOCK_NAME</td>
<td>7–20</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_BUGCHECK_MSG</td>
<td>7–22</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_CURRENT_CPU</td>
<td>7–24</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_CURRENT_PCB</td>
<td>7–25</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_HEADER</td>
<td>7–26</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_HW_NAME</td>
<td>7–28</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_IMAGE_OFFSET</td>
<td>7–29</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_INPUT</td>
<td>7–31</td>
<td></td>
</tr>
<tr>
<td>SDA$GET_LINE_COUNT</td>
<td>7–32</td>
<td></td>
</tr>
<tr>
<td>SDA$GETMEM</td>
<td>7–33</td>
<td></td>
</tr>
<tr>
<td>SDA$INSTRUCTION_DECODE</td>
<td>7–35</td>
<td></td>
</tr>
<tr>
<td>SDA$NEW_PAGE</td>
<td>7–37</td>
<td></td>
</tr>
<tr>
<td>SDA$PARSE_COMMAND</td>
<td>7–38</td>
<td></td>
</tr>
<tr>
<td>SDA$PRINT</td>
<td>7–40</td>
<td></td>
</tr>
<tr>
<td>SDA$READ_SYMFILE</td>
<td>7–42</td>
<td></td>
</tr>
<tr>
<td>SDA$REQMEM</td>
<td>7–44</td>
<td></td>
</tr>
<tr>
<td>SDA$SET_ADDRESS</td>
<td>7–46</td>
<td></td>
</tr>
<tr>
<td>SDA$SET_CPU</td>
<td>7–47</td>
<td></td>
</tr>
<tr>
<td>SDA$SET_HEADING_ROUTINE</td>
<td>7–48</td>
<td></td>
</tr>
<tr>
<td>SDA$SET_LINE_COUNT</td>
<td>7–50</td>
<td></td>
</tr>
<tr>
<td>SDA$SET_PROCESS</td>
<td>7–51</td>
<td></td>
</tr>
<tr>
<td>SDA$SKIP_LINES</td>
<td>7–52</td>
<td></td>
</tr>
<tr>
<td>SDA$SYMBOL_VALUE</td>
<td>7–53</td>
<td></td>
</tr>
<tr>
<td>SDA$SYMBOLIZE</td>
<td>7–54</td>
<td></td>
</tr>
<tr>
<td>SDA$TRYMEM</td>
<td>7–56</td>
<td></td>
</tr>
<tr>
<td>SDA$TYPE</td>
<td>7–58</td>
<td></td>
</tr>
<tr>
<td>SDA$VALIDATE_QUEUE</td>
<td>7–59</td>
<td></td>
</tr>
</tbody>
</table>

**Part II** OpenVMS Alpha System Code Debugger & System Dump Debugger
## 8 The OpenVMS Alpha System Code Debugger

8.1 User-Interface Options ........................................ 8–2
8.2 Building a System Image to Be Debugged ................... 8–2
8.3 Setting Up the Target System for Connections ............... 8–3
  8.3.1 Making Connections Between the Target Kernel and the System Code Debugger ............................................ 8–5
  8.3.2 Interactions Between XDELTA and the Target Kernel/System Code Debugger ............................................ 8–6
8.4 Setting Up the Host System .................................... 8–6
8.5 Starting the System Code Debugger ............................ 8–7
8.6 Summary of System Code Debugger Commands ............... 8–8
8.7 Using System Dump Analyzer Commands ...................... 8–8
8.8 System Code Debugger Network Information ................ 8–9
8.9 Troubleshooting Checklist .................................... 8–9
8.10 Troubleshooting Network Failures ............................ 8–10
8.11 Access to Symbols in OpenVMS Executive Images .......... 8–10
  8.11.1 Overview of How the OpenVMS Debugger Maintains Symbols .................................................. 8–10
8.11.2 Overview of OpenVMS Executive Image Symbols ........ 8–11
8.11.3 Possible Problems You May Encounter ................... 8–12
8.12 Sample System Code Debugging Session ..................... 8–14

## 9 The OpenVMS Alpha System Dump Debugger

9.1 User-Interface Options ........................................ 9–1
9.2 Preparing a System Dump to Be Analyzed ................... 9–2
9.3 Setting Up the Test System .................................... 9–3
9.4 Setting Up the Build System ................................... 9–3
9.5 Starting the System Dump Debugger ........................... 9–4
9.6 Summary of System Dump Debugger Commands ............... 9–4
9.7 Using System Dump Analyzer Commands ...................... 9–5
9.8 Limitations of the System Dump Debugger ................... 9–6
9.9 Access to Symbols in OpenVMS Executive Images .......... 9–6
9.10 Sample System Dump Debugging Session ..................... 9–6

## Part III OpenVMS Watchpoint Utility

### 10 The Watchpoint Utility

10.1 Introduction ................................................ 10–1
10.2 Initializing the Watchpoint Utility ............................ 10–2
10.3 Creating and Deleting Watchpoints ........................... 10–2
  10.3.1 Using the $QIO Interface ................................ 10–3
  10.3.2 Invoking WPDRIVER Entry Points from System Routines ................................................ 10–5
10.4 Data Structures ............................................ 10–6
  10.4.1 Watchpoint Restore Entry (WPRE) ..................... 10–6
  10.4.2 Watchpoint Control Blocks (WPCB) ..................... 10–6
  10.4.3 Trace Table Entries (WPTTEs) ......................... 10–7
10.5 Analyzing Watchpoint Results .............................. 10–7
10.6 Watchpoint Protection Overview ............................. 10–9
10.7 Restrictions .............................................. 10–10
Index

Examples
8–1 Booting the Target System ........................................ 8–14
8–2 Invoking the System Code Debugger .......................... 8–14
8–3 Connecting to the Target System ............................... 8–15
8–4 Target System Connection Display ............................ 8–16
8–5 Setting a Breakpoint ............................................. 8–16
8–6 Finding the Source Code ......................................... 8–17
8–7 Using the Set Mode Screen Command ......................... 8–18
8–8 Using the SCROLL/UP DEBUG Command ..................... 8–19
8–9 Breakpoint Display .................................................. 8–20
8–10 Using the Debug Step Command ............................... 8–21
8–11 Using the Examine and Show Calls Commands .......... 8–22
8–12 Canceling the Breakpoints ..................................... 8–23
8–13 Using the Step/Return Command ............................... 8–24
8–14 Source Lines Error Message .................................... 8–25
8–15 Using the Show Image Command .............................. 8–26
9–1 Invoking the System Dump Debugger ......................... 9–7
9–2 Accessing the System Dump ..................................... 9–7
9–3 Displaying the Source Code ..................................... 9–8
9–4 Using the Examine and Show Calls Commands .......... 9–9

Figures
2–1 Mechanism Array .................................................. 2–24
2–2 Signal Array ......................................................... 2–26
2–3 64-Bit Signal Array ................................................. 2–27
2–4 Exception Stack Frame ............................................ 2–28
2–5 Stack Following an Illegal Page-Fault Error ................ 2–33
8–1 Maintaining Symbols ............................................. 8–11
10–1 Format of Data Returned in Buffer .......................... 10–9

Tables
2–1 Definitions of Bits in DUMPSTYLE ............................. 2–4
2–2 Comparison of Full and Selective Dumps .................... 2–5
2–3 SDA Operators ....................................................... 2–15
2–4 Modules Containing Global Symbols and Data Structures Used by SDA ............................................. 2–18
2–5 SDA Symbols Defined on Initialization ......................... 2–18
2–6 SDA Symbols Defined by SET CPU Command ............... 2–19
2–7 SDA Symbols Defined by SET PROCESS Command ....... 2–19
2–8 Exception Stack Frame Values ................................... 2–28
4–1 Modules Defining Global Locations Within Executive Image 4–35
4–2 SET RMS Command Keywords for Displaying Process RMS Information ................................................................. 4–56
4–3 GSD Fields....................................................................................................................................................... 4–104
4–4 Contents of the SHOW LOCK and SHOW PROCESS/LOCKS Displays ................................................................. 4–118
4–5 Virtual Page Information in the SHOW PAGE_TABLE Display ........................................................................... 4–127
4–6 Type of Virtual Pages ........................................................................................................................................ 4–128
4–7 Bits In the PTE ................................................................................................................................................... 4–128
4–8 Physical Page Information in the SHOW PAGE_TABLE Display ........................................................................... 4–128
4–9 Types of Physical Pages ....................................................................................................................................... 4–129
4–10 Location of the Page ........................................................................................................................................... 4–129
4–11 Command Options with the /COLOR and /RADQualifiers .............................................................................. 4–135
4–12 Page Frame Number Information—Line One Fields ......................................................................................... 4–136
4–13 Page Frame Number Information—Line Two Fields ......................................................................................... 4–137
4–14 Flags Set in Page State ....................................................................................................................................... 4–137
4–15 /TYPE and /SUBTYPE Qualifier Examples ..................................................................................................... 4–142
4–16 Options for the /WORKING_SET_LIST Qualifier ............................................................................................. 4–156
4–17 Working Set List Entry Information in the SHOW PROCESS Display ........................................................................ 4–157
4–18 Process Section Table Entry Information in the SHOW PROCESS Display ............................................................. 4–158
4–19 Process I/O Channel Information in the SHOW PROCESS Display ........................................................................ 4–160
4–20 Image Information in the SHOW PROCESS Display ........................................................................................... 4–160
4–21 Resource Information in the SHOW RESOURCES Display ................................................................................... 4–176
4–22 Lock on Resources ................................................................................................................................................ 4–178
4–23 RMD Fields .......................................................................................................................................................... 4–180
4–24 Static Spinlocks .................................................................................................................................................... 4–191
4–25 Process Information in the SHOW SUMMARY Display .......................................................................................... 4–200
4–26 Current State Information .................................................................................................................................... 4–201
4–27 Options for the SHOW WORKING_SET_LIST Command ..................................................................................... 4–207
10–1 Driver Supported Functions .................................................................................................................................. 10–3
10–2 Returned Status Codes ........................................................................................................................................... 10–4
10–3 Returned Status Values ........................................................................................................................................... 10–5
Preface

Intended Audience

The OpenVMS Alpha System Analysis Tools Manual is intended primarily for the system programmer who must investigate the causes of system failures and debug kernel mode code, such as a device driver. This manual describes the following system analysis tools in detail; it also provides a summary of the dump off system disk (DOSD) feature and DELTA/XDELTA debugger:

- System Dump Analysis (SDA)
- System code debugger (SCD)
- System dump debugger (SDD)
- Watchpoint utility (WP)

This manual also includes such system management information as maintaining the system resources necessary to capture and store system crash dumps including the use of Dump off System Disk (DOSD). If you need to determine the cause of a hung process or improve system performance, refer to this manual for instructions on using the appropriate system analysis tool to analyze a running system.

Document Structure

The OpenVMS Alpha System Analysis Tools Manual includes the following information:

Chapter 1 presents an overview of all the system analysis tools. It describes the system dump analyzer (SDA), system code debugger (SCD), system dump debugger (SDD), and watchpoint utility (WP). It also provides a brief description of the dump off system disk (DOSD) feature and the DELTA/XDELTA debugger.

Part I describes the system dump analyzer (SDA) commands, SDA CLUE extension commands, and SDA extension commands.

Part II describes the system code debugger (SCD) and the system dump debugger (SDD).

Part III describes the Watchpoint utility (WP).

Related Documents

For additional information, refer to the following documents:

- OpenVMS Alpha Version 7.3 Upgrade and Installation Manual
- OpenVMS Calling Standard
- OpenVMS System Manager’s Manual, Volume 1: Essentials
• OpenVMS System Manager’s Manual, Volume 2: Tuning, Monitoring, and Complex Systems
• OpenVMS Programming Concepts Manual, Volume II
• Writing OpenVMS Alpha Device Drivers in C
• OpenVMS AXP Internals and Data Structures
• Alpha Architecture Reference Manual
• MACRO–64 Assembler for OpenVMS AXP Systems Reference Manual

For additional information about OpenVMS products and services, access the following World Wide Web address:

http://www.openvms.compaq.com/

Reader’s Comments

Compaq welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet  openvmsdoc@compaq.com
Mail      Compaq Computer Corporation
          OSSG Documentation Group, ZKO3-4/U08
          110 Spit Brook Rd.
          Nashua, NH 03062-2698

How To Order Additional Documentation

Use the following World Wide Web address to order additional documentation:

http://www.openvms.compaq.com/

If you need help deciding which documentation best meets your needs, call 800-282-6672.

Conventions

In this manual, any reference to OpenVMS is synonymous with Compaq OpenVMS.

VMScluster systems are now referred to as OpenVMS Cluster systems. Unless otherwise specified, references to OpenVMS Clusters or clusters in this document are synonymous with VMScusters.

The following conventions are used in this manual:

Ctrl/x A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.

PF1 x A sequence such as PF1 x indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.

In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)

In the HTML version of this document, this convention appears as brackets, rather than a box.
A horizontal ellipsis in examples indicates one of the following possibilities:

- Additional optional arguments in a statement have been omitted.
- The preceding item or items can be repeated one or more times.
- Additional parameters, values, or other information can be entered.

A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.

() In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.

[] In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.

| In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.

{} In command format descriptions, braces indicate a required choices; you must choose one of the options listed. Do not type the braces on the command line.

**bold text** This type face represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.

*italic text* Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error number), in command lines (/PRODUCER=name), and in command parameters in text (where dd represents the predefined code for the device type).

**UPPERCASE TEXT** Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.

**Monospace text** Monospace text indicates code examples and interactive screen displays.

In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.

- A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.

**numbers** All numbers in text are assumed to be hexadecimal unless otherwise noted. Other radices—binary, octal, or decimal—are explicitly indicated.
Overview of System Analysis Tools

This chapter presents an overview of the following system dump analysis tools and features:

- System Dump Analyzer (SDA)
- System Code Debugger (SCD)
- System Dump Debugger (SDD)
- Watchpoint Utility (WP)
- Delta/XDelta Debugger
- Dump Off System Disk (DOSD)

1.1 System Dump Analyzer (SDA)

The OpenVMS Alpha system dump analyzer (SDA) utility allows you to analyze a running system or a system dump after a system failure occurs. With a system failure, the operating system copies the contents of memory to a system dump file or the primary page file. Additionally, it records the hardware context of each processor. With SDA, you can interpret the contents of the dump file, examine the status of each processor at the time of the system failure, and investigate the possible causes of failure.

See Part I for complete information about SDA, SDA CLUE (Crash Log Utility Extractor), and SDA Extension routines.

1.2 System Code Debugger (SCD)

The OpenVMS Alpha System Code Debugger (SCD) allows you to debug non pageable system code and device drivers running at any interrupt priority level (IPL). You can use the SCD to perform the following tasks:

- Control the system software’s execution——stop at points of interest, resume execution, intercept fatal exceptions, and so on
- Trace the execution path of the system software
- Display the source code where the software is executing, and step by source line
- Monitor exception conditions
- Examine and modify the values of variables
- In some cases, test the effect of modifications without having to edit the source code, recompile, and relink

SCD is a symbolic debugger. You can specify variable names, routine names, and so on, precisely as they appear in your source code.
SCD recognizes the syntax, data typing, operators, expressions, scoping rules, and other constructs of a given language. If your code or driver is written in more than one language, you can change the debugging context from one language to another during a debugging session.

See Part II for complete information about SCD.

1.3 System Dump Debugger (SDD)

The OpenVMS Alpha System Dump Debugger allows you to analyze certain system dumps using the commands and semantics of SCD. You can use SDD to perform the following tasks:

- Display the source code where the software was executing at the time of the system failure
- Examine the values of variables and registers at the time of the system failure

SDD is a symbolic debugger. You can specify variable names, routine names, and so on, precisely as they appear in your source code.

SDD recognizes the syntax, data typing, operators, expressions, scoping rules, and other constructs of a given language. If your code or driver is written in more than one language, you can change the debugging context from one language to another during a debugging session.

See Part II for complete information about SDD.

1.4 Watchpoint Utility

The OpenVMS Watchpoint utility allows you to maintain a history of modifications that are made to a particular location in shared system space. It sets watchpoints on 32-bit and 64-bit addresses, and watches any system addresses whether in S0, S1, or S2 space.

See Part III for complete information about the Watchpoint utility.

1.5 Delta/XDelta Debugger

The OpenVMS Delta/XDelta debugger allows you to monitor the execution of user programs and the OpenVMS operating system. The Delta/XDelta debuggers both use the same commands and expressions, but they are different in how they operate. Delta operates as an exception handler in a process context; whereas XDelta is invoked directly from the hardware system control block (SCB) vector in a system context.

You use OpenVMS Delta instead of the OpenVMS symbolic debugger to debug programs that run in privileged processor mode at interrupt priority level (IPL) 0. Because Delta operates in a process context, you can use it to debug user-mode programs or programs that execute at interrupt priority level (IPL) 0 in any processor mode—user, supervisor, executive, and kernel. To run Delta in a processor mode other than user mode, your process must have the privilege that allows Delta to change to that mode: change-mode-to-executive (CMEXEC), or change-mode-to-kernel (CMKRNL) privilege. You cannot use Delta to debug code that executes at an elevated IPL. To debug with Delta, you invoke it from within your process by specifying it as the debugger instead of the symbolic debugger.
You use OpenVMS XDelta instead of the System Code Debugger when debugging system code that runs early in booting or when there is no Ethernet adaptor that can be dedicated to SCD. Because XDelta is invoked directly from the hardware system control block (SCB), it can be used to debug programs executing in any processor mode or at any IPL level. To use XDelta, you must have system privileges, and you must include XDelta when you boot the system. Since XDelta is not process specific, it is not invoked from a process. To debug with XDelta, you must boot the system with a command to include XDelta in memory. XDelta’s existence terminates when you reboot the system without XDelta.

On OpenVMS Alpha systems, XDelta supports 64-bit addressing. Quadword display mode displays full quadwords of information. The 64-bit address display mode accepts and displays all addresses as 64-bit quantities. XDelta has predefined command strings for displaying the contents of the page frame number (PFN) database.

You can use Delta/XDelta commands to perform the following debugging tasks:
- Open, display, and change the value of a particular location
- Set, clear, and display breakpoints
- Set, display modes in byte, word, longword, or ASCII
- Display instructions
- Execute the program in a single step with the option to step over a subroutine
- Set base registers
- List the names and locations of all loaded modules of the executive
- Map an address to an executive module

See the OpenVMS Delta/XDelta Debugger Manual for complete information about using the Delta/XDelta debugging utility.

1.6 Dump Off System Disk (DOSD)

The OpenVMS Alpha system allows you to write the system dump file to a device other than the system disk. This is useful in large memory systems and in clusters with common system disks where sufficient disk space, on one disk, is not always available to support your dump file requirements. To perform this activity, you must correctly enable the DUMPSTYLE system parameter to allow the bugcheck code to write the system dump file to an alternative device.

See the OpenVMS System Manager’s Manual, Volume 2: Tuning, Monitoring, and Complex Systems for complete information about how to write the system dump file to a disk other than the system disk.
Part I

OpenVMS Alpha System Dump Analyzer (SDA)

Part 1 describes the capabilities and system management of SDA. It provides how to use SDA by doing the following:

• Analyzing a system dump and a running system
• Understanding SDA context and commands
• Investigating system failures
• Inducing system failures
• Understanding the ANALYZE command and qualifiers
• Invoking SDA commands, SDA CLUE extension commands, SDA Spinlock Tracing commands, and SDA extension routines
This chapter describes the functions and the system management of SDA. It describes initialization, operation, and procedures in analyzing a system dump and analyzing a running system. This chapter also describes the SDA context, the command format, and the way both to investigate system failures and induce system failures.

2.1 Capabilities of SDA

When a system failure occurs, the operating system copies the contents of memory to a system dump file or the primary page file, recording the hardware context of each processor in the system as well. The System Dump Analyzer (SDA) is a utility that allows you to interpret the contents of this file, examine the status of each processor at the time of the system failure, and investigate the probable causes of the failure.

You can invoke SDA to analyze a system dump, using the DCL command ANALYZE/CRASH_DUMP. You can then use SDA commands to perform the following operations:

• Direct (or echo) the output of an SDA session to a file or device (SET OUTPUT or SET LOG).

• Display the condition of the operating system and the hardware context of each processor in the system at the time of the system failure (SHOW CRASH or CLUE CRASH).

• Select a specific processor in a multiprocessing system as the subject of analysis (SET CPU).

• Select the default size of address data manipulated by the EXAMINE and EVALUATE commands (SET FETCH).

• Enable or disable the sign extension of 32-bit addresses (SET SIGN_EXTEND).

• Display the contents of a specific process stack (SHOW STACK or CLUE STACK).

• Format a call frame from a stack location (SHOW CALL_FRAME).

• Read a set of global symbols into the SDA symbol table (READ).

• Define symbols to represent values or locations in memory and add them to the SDA symbol table (DEFINE).

• Delete symbols not required from the SDA symbol table (UNDEFINE).

• Evaluate an expression in hexadecimal and decimal, interpreting its value as a symbol, a condition value, a page table entry (PTE), a processor status (PS) quadword, or date and time (EVALUATE).
2.1 Capabilities of SDA

- Examine the contents of memory locations, optionally interpreting them as Alpha assembler instructions, a PTE, a PS, or date and time (EXAMINE).
- Display device status as reflected in system data structures (SHOW DEVICE).
- Display the contents of the stored machine check frame (SHOW MACHINE_CHECK or CLUE MCHK) for selected Compaq computers.
- Format system data structures (FORMAT).
- Validate the integrity of the links in a queue (VALIDATE QUEUE).
- Display a summary of all processes on the system (SHOW SUMMARY).
- Show the hardware or software context of a process (SHOW PROCESS or CLUE PROCESS).
- Display the OpenVMS RMS data structures of a process (SHOW PROCESS with the /RMS qualifier).
- Display memory management data structures (SHOW POOL, SHOW PFN_DATA, SHOW PAGE_TABLE, or CLUE MEMORY).
- Display lock management data structures (SHOW RESOURCE or SHOW LOCK).
- Display OpenVMS Cluster management data structures (SHOW CLUSTER, SHOW CONNECTIONS, SHOW RSPID, or SHOW PORTS).
- Display multiprocessor synchronization information (SHOW SPINLOCKS).
- Display the layout of the executive images (SHOW EXECUTIVE).
- Capture and archive a summary of dump file information in a list file (CLUE HISTORY).
- Copy the system dump file (COPY).
- Define keys to invoke SDA commands (DEFINE/KEY).
- Search memory for a given value (SEARCH).

Although SDA provides a great deal of information, it does not automatically analyze all the control blocks and data contained in memory. For this reason, in the event of system failure, it is extremely important that you save not only the output provided by SDA commands, but also a copy of the system dump file written at the time of the failure.

You can also invoke SDA to analyze a running system, using the DCL command ANALYZE/SYSTEM. Most SDA commands generate useful output when entered on a running system.

**Caution:**

Although analyzing a running system may be instructive, you should undertake such an operation with caution. System context, process context, and a processor's hardware context can change during any given display.

In a multiprocessing environment, it is very possible that, during analysis, a process running SDA could be rescheduled to a different processor frequently. Therefore, avoid examining the hardware context of processors in a running system.
2.2 System Management and SDA

The system manager must ensure that the system writes a dump file whenever the system fails. The manager must also see that the dump file is large enough to contain all the information to be saved, and that the dump file is saved for analysis. The following sections describe these tasks.

2.2.1 Writing System Dumps

The operating system attempts to write information into the system dump file only if the system parameter DUMPBUG is set. (The DUMPBUG parameter is set by default. To examine and change its value, consult the OpenVMS System Manager’s Manual, Volume 2: Tuning, Monitoring, and Complex Systems.) If DUMPBUG is set and the operating system fails, the system manager has the following choices for writing system dumps:

- Have the system dump file written to either SYSDUMP.DMP (the system dump file) or to PAGEFILE.SYS (the primary system page file).
- Set the DUMPSTYLE system parameter to an even number (for dumps containing all physical memory) or to an odd number (for dumps containing only selected virtual addresses). See Section 2.2.1.1 for more information about the DUMPSTYLE parameter values.

2.2.1.1 Dump File Style

There are two types of dump files—a full memory dump (also known as a physical dump), and a dump of selected virtual addresses (also known as a selective dump). Both full and selective dumps may be produced in either compressed or uncompressed form. Compressed dumps save disk space and time taken to write the dump at the expense of a slight increase in time to access the dump with SDA. The SDA commands COPY/COMPRESS and COPY/DECOMPRESS can be used to convert an existing dump.

A dump can be written to the system disk, or to another disk set aside for dumps. When using a disk other than a system disk, the disk name is set in the console environment variable DUMP_DEV. This disk is also known as the “dump off system disk” (DOSD) disk.

When writing a system dump, information about the crash is displayed at the system console. This can be either minimal output (for example, bug check code, process name, and image name), or verbose output (for example, executive layout, stack and register contents).

In an OpenVMS Alpha Galaxy system, shared memory is dumped by default. It is sometimes necessary to disable the dumping of shared memory. For more information about shared memory, see OpenVMS Alpha Galaxy Guide.

DUMPSTYLE, which specifies the method of writing system dumps, is a 32-bit mask. Table 2–1 shows how the bits are defined. Each bit can be set independently. The value of the SYSGEN parameter is the sum of the values of the bits that have been set. Remaining or undefined values are reserved to Compaq.
Table 2–1 Definitions of Bits in DUMPSTYLE

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0   | 1     | 0 = Full dump. The entire contents of physical memory will be written to the dump file.  
1 = Selective dump. The contents of memory will be written to the dump file selectively to maximize the usefulness of the dump file while conserving disk space. (Only pages that are in use are written). |
| 1   | 2     | 0 = Minimal console output. This consists of the bugcheck code; the identity of the CPU, process, and image where the crash occurred; the system date and time; plus a series of dots indicating progress writing the dump.  
1 = Full console output. This includes the minimal output previously described plus stack and register contents, system layout, and additional progress information such as the names of processes as they are dumped. |
| 2   | 4     | 0 = Dump to system disk. The dump will be written to SYS$SYSDEVICE:[SYSn.SYSEXE]SYSDUMP.DMP, or in its absence, SYS$SYSDEVICE:[SYSn.SYSEXE]PAGEFILE.SYS.  
1 = Dump to alternate disk. The dump will be written to dump_dev:[SYSn.SYSEXE]SYSDUMP.DMP, where dump_dev is the value of the console environment variable DUMP_DEV. |
| 3   | 8     | 0 = Uncompressed dump. Pages are written directly to the dump file.  
1 = Compressed dump. Each page is compressed before it is written, providing a saving in space and in the time taken to write the dump, at the expense of a slight increase in time taken to access the dump. |
| 4   | 16    | 0 = Dump shared memory.  
1 = Do not dump shared memory. |
| 5–31|       | Reserved to Compaq |

The default setting for DUMPSTYLE is 0 (an uncompressed full dump, including shared memory, written to the system disk). Unless a value for DUMPSTYLE is specified in MODPARAMS.DAT, AUTGEN.COM will set DUMPSTYLE either to 1 (an uncompressed selective dump, including shared memory, written to the system disk) if there is less than 128 megabytes of memory on the system, or to 9 (a compressed selective dump, including shared memory, written to the system disk).

2.2.1.2 Comparison of Full and Selective Dumps

A full dump requires that all physical memory be written to the dump file. This ensures the presence of all the page table pages required for SDA to emulate translation of system virtual addresses. Any even-numbered value in the DUMPSTYLE system parameter generates a full dump.

In certain system configurations, it may be impossible to preserve the entire contents of memory in a disk file. For instance, a large memory system or a system with small disk capacity may not be able to supply enough disk space for a full memory dump. If the system dump file cannot accommodate all of memory, information essential to determining the cause of the system failure may be lost.
To preserve those portions of memory that contain information most useful in determining the causes of system failures, a system manager sets the value of the DUMPSTYLE system parameter to specify a dump of selected virtual address spaces. In a selective dump, related pages of virtual address space are written to the dump file as units called logical memory blocks (LMBs). For example, one LMB consists of the page tables for system space; another is the address space of a particular process. Those LMBs most likely to be useful in crash dump analysis are written first. Any odd-numbered value in the DUMPSTYLE system parameter generates a selective dump.

Table 2–2 compares full and selective style dumps.

<table>
<thead>
<tr>
<th>Item</th>
<th>Full</th>
<th>Selective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Information</td>
<td>Complete contents of physical memory in use, stored in order of increasing physical address.</td>
<td>System page table, global page table, system space memory, and process and control regions (plus global pages) for all saved processes.</td>
</tr>
<tr>
<td>Unavailable Information</td>
<td>Contents of paged-out memory at the time of the system failure.</td>
<td>Contents of paged-out memory at the time of the system failure, process and control regions of unsaved processes, and memory not mapped by a page table.</td>
</tr>
<tr>
<td>SDA Command Limitations</td>
<td>None.</td>
<td>The following commands are not useful for unsaved processes: SHOW PROCESS/CHANNELS, SHOW PROCESS/IMAGE, SHOW PROCESS/RMS, SHOW STACK, and SHOW SUMMARY/IMAGE.</td>
</tr>
</tbody>
</table>

2.2.1.3 Controlling the Size of Page Files and Dump Files

You can adjust the size of the system page file and dump file using AUTOGEN (the recommended method) or by using SYSGEN.

AUTOGEN automatically calculates the appropriate sizes for page and dump files. AUTOGEN invokes the System Generation utility (SYSGEN) to create or change the files. However, you can control sizes calculated by AUTOGEN by defining symbols in the MODPARAMS.DAT file. The file sizes specified in MODPARAMS.DAT are copied into the PARAMS.DAT file during AUTOGEN’s GETDATA phase. AUTOGEN then makes appropriate adjustments in its calculations.

Although Compaq recommends using AUTOGEN to create and modify page and dump file sizes, you can use SYSGEN to directly create and change the sizes of those files.

The sections that follow discuss how you can calculate the size of a dump file.

See the OpenVMS System Manager’s Manual for detailed information about using AUTOGEN and SYSGEN to create and modify page and dump file sizes.

2.2.1.4 Writing to the System Dump File

OpenVMS Alpha writes the contents of the error-log buffers, processor registers, and memory into the system dump file, overwriting its previous contents. If the system dump file is too small, OpenVMS Alpha cannot copy all memory to the file when a system failure occurs.
2.2 System Management and SDA

SYS$SYSTEM:SYSDUMP.DMP (SYS$SPECIFIC:[SYSEXE]SYSDUMP.DMP) is created during installation. To successfully store a crash dump, SYS$SYSTEM:SYSDUMP.DMP must be enlarged to hold all of memory (full dump) or all of system space and the key processes (selective dump).

To calculate the correct size for an uncompressed full dump to SYS$SYSTEM:SYSDUMP.DMP, use the following formula:

\[
\text{size-in-blocks} = \text{size-in-pages(physical-memory)} \times \text{blocks-per-page} \\
+ \text{number-of-error-log-buffers} \times \text{blocks-per-buffer} \\
+ 10
\]

Use the DCL command SHOW MEMORY to determine the total size of physical memory on your system. There is a variable number of error log buffers in any given system, depending on the setting of the ERRORLOGBUFFERS system parameter. The size of each buffer depends on the setting of the ERLBUFFERPAGES parameter. (See the OpenVMS System Manager’s Manual for additional information about these parameters.)

2.2.1.5 Writing to the Dump File off the System Disk

OpenVMS Alpha allows you to write the system dump file to a device other than the system disk. This is useful in large memory systems and in clusters with common system disks where sufficient disk space, on one disk, is not always available to support customer dump file requirements. To perform this activity, the DUMPSTYLE system parameter must be correctly enabled to allow the bugcheck code to write the system dump file to an alternative device.

The requirements for writing the system dump file off the system disk are the following:

- The dump device directory structure must resemble the current system disk structure. The [SYSn.SYSEXE]SYSDUMP.DMP file will reside there, with the same boot time system root.
  
  You can use AUTOGEN to create this file. In the MODPARAMS.DAT file, the following symbol prompts AUTOGEN to create the file:

  \[
  \text{DUMPFILEDEVICE} = \text{$nnn$ddcuuuu}
  \]

- The dump device cannot be part of a volume set or a member of a shadow set.
- You must set DOSD for SDA CLUE as described in Chapter 5.
- The DUMP_DEV environment variable must exist on your system. You specify the dump device at the console prompt, using the following format:

  \[
  \text{SET DUMP_DEV device-name[,..]}
  \]

  On some CPU types, you can enter a list of devices. The list can include various alternate paths to the system disk and the dump disk.

  By specifying alternate paths in DUMP_DEV, a dump can still be written if the disk fails over to an alternate path while the system is running. When the system crashes, the bugcheck code can use the alternate path by referring to the contents of DUMP_DEV.

  When you enter a list of devices, however, the system disk must come last.

For information on how to write the system dump file to an alternative device to the system disk, see the OpenVMS System Manager’s Manual, Volume 2: Tuning, Monitoring, and Complex Systems.
2.2.1.6 Writing to the System Page File

If SYSSSYSTEM:SYSDUMP.DMP does not exist, and there is no DOSD device or dump file, the operating system writes the dump of physical memory into SYSSSYSTEM:PAGEFILE.SYS, the primary system page file, overwriting the contents of that file.

If the SAVEDUMP system parameter is set, the dump file is retained in PAGEFILE.SYS when the system is booted after a system failure. If the SAVEDUMP parameter is not set, which is the default, OpenVMS Alpha uses the entire page file for paging and any dump written to the page file is lost. (To examine or change the value of the SAVEDUMP parameter, consult the OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems.)

To calculate the minimum size for a full memory dump to SYSSSYSTEM:PAGEFILE.SYS, use the following formula:

\[
\text{size-in-blocks}(\text{SYSSSYSTEM:PAGEFILE.SYS}) = \text{size-in-pages(physical-memory)} \times \text{blocks-per-page} \\
+ \text{number-of-error-log-buffers} \times \text{blocks-per-buffer} \\
+ 10 \\
+ \text{value of the system parameter RSRVPAGCNT} \times \text{blocks-per-page}
\]

Note that this formula calculates the minimum size requirement for saving a physical dump in the system's page file. Compaq recommends that the page file be a bit larger than this minimum to avoid hanging the system. Also note that you can only write the system dump into the primary page file (SYSSSYSTEM:PAGEFILE.SYS). Secondary page files cannot be used to save dump file information.

Note also that OpenVMS will not fill the page file completely when writing a system dump, since the system might hang when rebooting after a system crash. RSRVPAGCNT pages are kept unavailable for dumps. This applies to both full dumps and selective dumps.

Writing crash dumps to SYSSSYSTEM:PAGEFILE.SYS presumes that you will later free the space occupied by the dump for use by the pager. Otherwise, your system may hang during the startup procedure. To free this space, you can do one of the following:

- Include SDA commands that free dump space in the site-specific startup command procedure (described in Section 2.2.3).
- Use the SDA COPY command to copy the dump from SYSSSYSTEM:PAGEFILE.SYS to another file. Use the SDA COPY command instead of the DCL COPY command because the SDA COPY command only copies the blocks used by the dump and causes the pages occupied by the dump to be freed from the system's page file.
- If you do not need to copy the dump elsewhere, issue an ANALYZE/CRASH_DUMP/RELEASE command. When you issue this command, SDA immediately releases the pages to be used for system paging, effectively deleting the dump. Note that this command does not allow you to analyze the dump before deleting it.
2.2.2 Saving System Dumps

Every time the operating system writes information to the system dump file, it writes over whatever was previously stored in the file. The system writes information to the dump file whenever the system fails. For this reason, the system manager must save the contents of the file after a system failure has occurred.

The system manager can use the SDA COPY command or the DCL COPY command. Either command can be used in a site-specific startup procedure, but the SDA COPY command is preferred because it marks the dump file as copied. As mentioned earlier, this is particularly important if the dump was written into the page file, SYS$SYSTEM:PAGEFILE.SYS, because it releases those pages occupied by the dump to the pager. Another advantage of using the SDA COPY command is that this command copies only the saved number of blocks and not necessarily the whole allotted dump file. For instance, if the size of the SYSDUMP.DMP file is 100,000 blocks and the bugcheck wrote only 60,000 blocks to the dump file, then DCL COPY would create a file of 100,000 blocks. However, SDA COPY would generate a file of only 60,000 blocks.

Because system dump files are set to NOBACKUP, the Backup utility (BACKUP) does not copy them to tape unless you use the qualifier /IGNORE=NOBACKUP when invoking BACKUP. When you use the SDA COPY command to copy the system dump file to another file, OpenVMS Alpha does not set the new file to NOBACKUP.

As created during installation, the file SYS$SYSTEM:SYSDUMP.DMP is protected against world access. Because a dump file can contain privileged information, Compaq recommends that the system manager does not change this default protection.

2.2.3 Invoking SDA When Rebooting the System

When the system reboots after a system failure, SDA is automatically invoked by default. SDA archives information from the dump in a history file. In addition, a listing file with more detailed information about the system failure is created in the directory pointed to by the logical name CLUE$COLLECT. (Note that the default directory is SYS$ERRORLOG unless you redefine the logical name CLUE$COLLECT in the procedure SYSSMANAGER:SYLOGICALS.COM.) The file name is in the form CLUE$node_ddmmyy_hhmm.LIS where the timestamp (hhmm) corresponds to the system failure time and not the time when the file was created.

Directed by commands in a site-specific file, SDA can take additional steps to record information about the system failure. They include the following:

- Copying the contents of the dump file to another file. This information is otherwise lost at the next system failure when the system saves information only about that failure.
- Supplementing the contents of the list file containing the output of specific SDA commands.

If the logical name CLUE$SITE_PROC points to a valid and existing command file, it will be executed as part of the CLUE HISTORY command when you reboot. If used, this file should contain only valid SDA commands.
2.2 System Management and SDA

Generated by a set sequence of commands, the CLUE list file contains only an overview of the failure and is unlikely to provide enough information to determine the cause of the failure. Compaq, therefore, recommends that you always copy the dump file.

The following example shows SDA commands that can make up your site-specific command file to produce a more complete SDA listing after each system failure, and to save a copy of the dump file:

```
! SDA command file, to be executed as part of the system
! bootstrap from within CLUE. Commands in this file can
! be used to save the dump file after a system bugcheck, and
! to execute any additional SDA commands.
!
! Note that the logical name DMP$ must have been defined
! within SYS$MANAGER:SYLOGICALS.COM
!
READ/EXEC       ! read in the executive images’ symbol tables
SHOW STACK      ! display the stack
COPY DMP$:SAVEDUMP.DMP  ! copy and save dump file
```

The CLUE HISTORY command is executed first, followed by the SDA commands in this site-specific command file. See the reference section on CLUE HISTORY for details on the summary information that is generated and stored in the CLUE list file by the CLUE HISTORY command. Note that the SDA COPY command is final command. If the dump has been written to PAGEFILE.SYS, then the space used by the dump will be automatically returned for use for paging as soon as the COPY is complete and no more analysis is possible.

To point to your site-specific file, add a line such as the following to the file SYS$MANAGER:SYLOGICALS.COM:

```
$ DEFINE/SYSTEM CLUES$SITE_PROC SYS$MANAGER:SAVEDUMP.COM
```

In this example, the site-specific file is named SAVEDUMP.COM.

The CLUE list file can be printed immediately or saved for later examination.

SDA is invoked and executes the specified commands only when the system boots for the first time after a system failure. If the system is booting for any other reason (such as a normal system shutdown and reboot), SDA exits.

If CLUE files occupy more space than the threshold allows (the default is 5000 blocks), the oldest files will be deleted until the threshold limit is reached. The threshold limit can be customized with the CLUE$MAX_BLOCK logical name.

To prevent the running of CLUE at system startup, define the logical CLUE$INHIBIT in the SYLOGICALS.COM file as TRUE in the system logical name table.

### 2.3 Analyzing a System Dump

SDA performs certain tasks before bringing a dump into memory, presenting its initial displays, and accepting command input. These tasks include the following:

- Verifying that the process invoking it is suitably privileged to read the dump file
- Using RMS to read in pages from the dump file
2.3 Analyzing a System Dump

- Building the SDA symbol table from the files SDA$READ_DIR:SYS$BASE_IMAGE.EXE and SDA$READ_DIR:REQSYSDEF.STB
- Executing the commands in the SDA initialization file

For detailed information on investigating system failures, see Section 2.7.

2.3.1 Requirements

To analyze a dump file, your process must have read access both to the file that contains the dump and to copies of SDA$READ_DIR:SYS$BASE_IMAGE.EXE and SDA$READ_DIR:REQSYSDEF.STB (the required subset of the symbols in the file SYSDEF.STB). SDA reads these tables by default.

2.3.2 Invoking SDA

If your process can access the files listed in Section 2.3.1, you can issue the DCL command ANALYZE/CRASH_DUMP to invoke SDA. If you do not specify the name of a dump file in the command, SDA prompts you:

$ ANALYZE/CRASH_DUMP
_Dump File:

The default file specification is as follows:

SYS$DISK:[default-dir]SYSDUMP.DMP

SYS$DISK and [default-dir] represent the disk and directory specified in your last SET DEFAULT command.

If you are rebooting after a system failure, SDA is automatically invoked. See Section 2.2.3.

2.3.3 Mapping the Contents of the Dump File

SDA first attempts to map the contents of memory as stored in the specified dump file. To do this, it must first locate the page tables for system space among its contents. The system page tables contain one entry for each page of system virtual address space.

- If SDA cannot find the system page tables in the dump file, it displays the following message:

  %SDA-E-SPTNOTFND, system page table not found in dump file

  If that error message is displayed, you cannot analyze the crash dump, but must take steps to ensure that any subsequent dump can be analyzed. To do this, you must either adjust the DUMPSTYLE system parameter as discussed in Section 2.2.1.1 or increase the size of the dump file as indicated in Section 2.2.1.3.

- If SDA finds the system page tables in an incomplete dump, the following message is displayed:

  %SDA-W-SHORTDUMP, dump file was n blocks too small when dump written; analysis may not be possible

Under certain conditions, some memory locations might not be saved in the system dump file. Additionally, if a bugcheck occurs during system initialization, the contents of the register display may be unreliable. The symptom of such a bugcheck is a SHOW SUMMARY display that shows no processes or only the swapper process.
If you use an SDA command to access a virtual address that has no corresponding physical address, SDA generates the following error message:

%SDA-E-NOTINPHYS, ‘location’: virtual data not in physical memory

When analyzing a selective dump file, if you use an SDA command to access a virtual address that has a corresponding physical address not saved in the dump file, SDA generates one of the following error messages:

%SDA-E-MEMNOTSVD, memory not saved in the dump file
%SDA-E-NOREAD, unable to access location n

2.3.4 Building the SDA Symbol Table

After locating and reading the system dump file, SDA attempts to read the system symbol table file into the SDA symbol table. If SDA cannot find SDA$READ_DIR:SYS$BASE_IMAGE.EXE—or is given a file that is not a system symbol table in the /SYMBOL qualifier to the ANALYZE command—it displays a fatal error and exits. SDA also reads into its symbol table a subset of SDA$READ_DIR:SYSDEF.STB, called SDA$READ_DIR:REQSYSDEF.STB. This subset provides SDA with the information needed to access some of the data structures in the dump.

When SDA finishes building its symbol table, SDA displays a message identifying itself and the immediate cause of the system failure. In the following example, the cause of the system failure was the deallocation of a bad page file address.

OpenVMS Alpha System Dump Analyzer
BADPAGFILD, Bad page file address deallocated

2.3.5 Executing the SDA Initialization File (SDA$INIT)

After displaying the system failure summary, SDA executes the commands in the SDA initialization file, if you have established one. SDA refers to its initialization file by using the logical name SDA$INIT. If SDA cannot find the file defined as SDA$INIT, it searches for the file SYS$LOGIN:SDA.INIT.

This initialization file can contain SDA commands that read symbols into SDA's symbol table, define keys, establish a log of SDA commands and output, or perform other tasks. For instance, you may want to use an SDA initialization file to augment SDA's symbol table with definitions helpful in locating system code. If you issue the following command, SDA includes those symbols that define many of the system's data structures, including those in the I/O database:

READ SDA$READ_DIR:filename

You may also find it helpful to define those symbols that identify the modules in the images that make up the executive by issuing the following command:

READ/EXECUTIVE SDA$READ_DIR:

After SDA has executed the commands in the initialization file, it displays its prompt as follows:

SDA>

This prompt indicates that you can use SDA interactively and enter SDA commands.

An SDA initialization file may invoke a command procedure with the @command. However, such command procedures cannot invoke other command procedures.
Occasionally, OpenVMS Alpha encounters an internal problem that hinders system performance without causing a system failure. By allowing you to examine the running system, SDA enables you to search for the solution without disturbing the operating system. For example, you may be able to use SDA to examine the stack and memory of a process that is stalled in a scheduler state, such as a miscellaneous wait (MWAIT) or a suspended (SUSP) state.

If your process has change-mode-to-kernel (CMKRNL) privilege, you can invoke SDA to examine the system. Use the following DCL command:

```
$ ANALYZE/SYSTEM
```

SDA attempts to load SDA$READ_DIR:SYS$BASE_IMAGE.EXE and SDA$READ_DIR:REQSYSDEF.STB. It then executes the contents of any existing SDA initialization file, as it does when invoked to analyze a crash dump (see Sections 2.3.4 and 2.3.5, respectively). SDA subsequently displays its identification message and prompt, as follows:

```
OpenVMS Alpha System Analyzer
SDA>
```

This prompt indicates that you can use SDA interactively and enter SDA commands. When analyzing a running system, SDA sets its process context to that of the process running SDA.

If you are analyzing a running system, consider the following:

- When used in this mode, SDA does not map the entire system, but instead retrieves only the information it needs to process each individual command. To update any given display, you must reissue the previous command.

---

**Caution:**

When using SDA to analyze a running system, carefully interpret its displays. Because system states change frequently, it is possible that the information SDA displays may be inconsistent with the current state of the system.

---

- Certain SDA commands are illegal in this mode, such as SHOW CPU and SET CPU. Use of these commands results in the following error message:

  ```
  %SDA-E-CMDNOTVLD, command not valid on the running system
  ```

- The SHOW CRASH command, although valid, does not display the contents of any of the processor’s set of hardware registers.

### 2.5 SDA Context

When you invoke SDA to analyze either a crash dump or a running system, SDA establishes a default context for itself from which it interprets certain commands.

When you are analyzing a uniprocessor system, SDA's context is solely **process context**, which means SDA can interpret its process-specific commands in the context of either the process current on the uniprocessor or some other process in another scheduling state. When SDA is initially invoked to analyze a crash dump, SDA’s process context defaults to that of the process that was current at the time of the system failure. When you invoke SDA to analyze a running system, SDA sets its process context to that of the process running SDA.

SDA subsequently displays its identification message and prompt, as follows:

```
OpenVMS Alpha System Analyzer
SDA>
```

This prompt indicates that you can use SDA interactively and enter SDA commands. When analyzing a running system, SDA sets its process context to that of the process running SDA.

If you are analyzing a running system, consider the following:

- When used in this mode, SDA does not map the entire system, but instead retrieves only the information it needs to process each individual command. To update any given display, you must reissue the previous command.

---

**Caution:**

When using SDA to analyze a running system, carefully interpret its displays. Because system states change frequently, it is possible that the information SDA displays may be inconsistent with the current state of the system.

---

- Certain SDA commands are illegal in this mode, such as SHOW CPU and SET CPU. Use of these commands results in the following error message:

  ```
  %SDA-E-CMDNOTVLD, command not valid on the running system
  ```

- The SHOW CRASH command, although valid, does not display the contents of any of the processor’s set of hardware registers.
system, SDA's process context defaults to that of the current process, that is, the one executing SDA. To change SDA's process context, issue any of the following commands:

- `SET PROCESS process-name`
- `SET PROCESS/ADDRESS=pcb-address`
- `SET PROCESS/INDEX=nn`
- `SET PROCESS/SYSTEM`
- `SHOW PROCESS process-name`
- `SHOW PROCESS/ADDRESS=pcb-address`
- `SHOW PROCESS/INDEX=nn`
- `SHOW PROCESS/SYSTEM`

When you invoke SDA to analyze a crash dump from a multiprocessing system with more than one active CPU, SDA maintains a second dimension of context—its CPU context—that allows it to display certain processor-specific information. This information includes the reason for the bugcheck exception, the currently executing process, the current IPL, and the spin locks owned by the processor.

When you invoke SDA to analyze a multiprocessor's crash dump, its CPU context defaults to that of the processor that induced the system failure. When you are analyzing a running system, CPU context is not accessible to SDA. Therefore, the SET CPU and SHOW CPU commands are not permitted.

You can change the SDA CPU context by using any of the following commands:

- `SET CPU cpu-id`
- `SHOW CPU cpu-id`
- `SHOW CRASH`
- `SHOW MACHINE_CHECK cpu-id`

Changing CPU context involves an implicit change in process context in either of the following ways:

- If there is a current process on the CPU made current, SDA process context is changed to that of that CPU's current process.

- If there is no current process on the CPU made current, SDA process context is undefined and no process-specific information is available until SDA process context is set to that of a specific process.

Changing process context can require a switch of CPU context as well. For instance, if you issue a SET PROCESS command for a process that was current at the time of a system failure on another CPU, SDA will automatically change its CPU context to that of the CPU on which that process was current. The following commands can have this effect if the process-name, pcb-address, or index number (nn) refers to a current process:

- `SET PROCESS process-name`
- `SET PROCESS/ADDRESS=pcb-address`
- `SET PROCESS/INDEX=nn`
- `SHOW PROCESS process-name`
- `SHOW PROCESS/ADDRESS=pcb-address`
- `SHOW PROCESS/INDEX=nn`
2.6 SDA Command Format

The following sections describe the format of SDA commands and the expressions you can use with SDA commands.

2.6.1 General Command Format

SDA uses a command format similar to that used by the DCL interpreter. Issue commands in the following format:

`command-name[/qualifier...] [parameter][/qualifier...] [!comment]`

The **command-name** is an SDA command. Each command tells the utility to perform a function. Commands can consist of one or more words, and can be abbreviated to the number of characters that make the command unique. For example, SH stands for SHOW.

The **parameter** is the target of the command. For example, SHOW PROCESS RUSKIN tells SDA to display the context of the process RUSKIN. The command EXAMINE 80104CD0;40 displays the contents of 40 bytes of memory, beginning with location 80104CD0.

When you supply part of a file specification as a parameter, SDA assumes default values for the omitted portions of the specification. The default device is SYS$DISK, the device specified in your most recent SET DEFAULT command. The default directory is the directory specified in the most recent SET DEFAULT command. See the OpenVMS DCL Dictionary for a description of the DCL command SET DEFAULT.

The **qualifier** modifies the action of an SDA command. A qualifier is always preceded by a slash (/). Several qualifiers can follow a single parameter or command name, but each must be preceded by a slash. Qualifiers can be abbreviated to the shortest string of characters that uniquely identifies the qualifier.

The **comment** consists of text that describes the command; this comment is not actually part of the command. Comments are useful for documenting SDA command procedures. When executing a command, SDA ignores the exclamation point and all characters that follow it on the same line.

2.6.2 Expressions

You can use expressions as parameters for some SDA commands, such as SEARCH and EXAMINE. To create expressions, use any of the following elements:

- Numerals
- Radix operators
- Arithmetic and logical operators
- Precedence operators
- Symbols

Numerals are one possible component of an expression. The following sections describe the use of the other components.
2.6.2.1 Radix Operators

Radix operators determine which numeric base SDA uses to evaluate expressions. You can use one of the three radix operators to specify the radix of the numeric expression that follows the operator:

- `^X` (hexadecimal)
- `^O` (octal)
- `^D` (decimal)

The default radix is hexadecimal. SDA displays hexadecimal numbers with leading zeros and decimal numbers with leading spaces.

2.6.2.2 Arithmetic and Logical Operators

There are two types of arithmetic and logical operators, both of which are listed in Table 2–3.

- **Unary operators** affect the value of the expression that follows them.
- **Binary operators** combine the operands that precede and follow them.

In evaluating expressions containing binary operators, SDA performs logical AND, OR, and XOR operations, and multiplication, division, and arithmetic shifting before addition and subtraction. Note that the SDA arithmetic operators perform integer arithmetic on 64-bit operands.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#</code></td>
<td>Performs a logical NOT of the expression</td>
</tr>
<tr>
<td><code>+</code></td>
<td>Makes the value of the expression positive</td>
</tr>
<tr>
<td><code>-</code></td>
<td>Makes the value of the expression negative</td>
</tr>
<tr>
<td><code>@</code></td>
<td>Evaluates the following expression as an address, then uses the contents of that address as value</td>
</tr>
<tr>
<td><code>^Q</code></td>
<td>Specifies that the size of field to be used as an address is a quadword when used with the unary operator @ (^1)</td>
</tr>
<tr>
<td><code>^L</code></td>
<td>Specifies that the size of field to be used as an address is a longword when used with the unary operator @ (^2)</td>
</tr>
<tr>
<td><code>^W</code></td>
<td>Specifies that the size of field to be used as an address is a word when used with the unary operator @ (^3)</td>
</tr>
<tr>
<td><code>^B</code></td>
<td>Specifies that the size of field to be used as an address is a byte when used with the unary operator @ (^4)</td>
</tr>
<tr>
<td><code>^P</code></td>
<td>Specifies a physical address when used with the unary operator @ (^5)</td>
</tr>
<tr>
<td><code>^V</code></td>
<td>Specifies a virtual address when used with the unary operator @ (^6)</td>
</tr>
<tr>
<td><code>G</code></td>
<td>Adds FFFFFFFF 80000000 16 to the value of the expression (^2)</td>
</tr>
</tbody>
</table>

\(^1\)The command SET FETCH can be used to change the default FETCH size and/or access method. See the SET FETCH command description in Chapter 4 for more details and examples.

\(^2\)The unary operator G corresponds to the first virtual address in system space. For example, the expression GD40 can be used to represent the address FFFFFFFF 80000D40 16.

(continued on next page)
Table 2–3 (Cont.) SDA Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unary Operators</strong></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>Adds $7ffe0000_{16}$ to the value of the expression.</td>
</tr>
<tr>
<td>I</td>
<td>Fills the leading digits of the following hexadecimal number with hex value of F. For example:</td>
</tr>
<tr>
<td></td>
<td>SDA&gt; eval i80000000</td>
</tr>
<tr>
<td></td>
<td>Hex = FFFFFFFF.80000000 Decimal = -2147483648</td>
</tr>
</tbody>
</table>

| **Binary Operators** | |
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| & | Logical AND |
| | Logical OR |
| \ | Logical XOR |
| / | Division |
| @ | Arithmetic shifting |
| "." | Catenates two 32-bit values into a 64-bit value. For example: |
| | SDA> eval fe.50000 |
| | Hex = 000000FE00050000 Decimal = 1090922020864 |

3 The unary operator H corresponds to a convenient base address in P1 space ($7ffe0000_{16}$). You can therefore refer to an address such as $7ffe2a64_{16}$ as H2A64.

4 In division, SDA truncates the quotient to an integer, if necessary, and does not retain a remainder.

2.6.2.3 Precedence Operators

SDA uses parentheses as precedence operators. Expressions enclosed in parentheses are evaluated first. SDA evaluates nested parenthetical expressions from the innermost to the outermost pairs of parentheses.

2.6.2.4 Symbols

A symbol can represent a few different types of values. It can represent a constant, a data address, a procedure descriptor address, or a routine address. Constants are usually offsets of a particular field in a data structure; however, they can also represent constant values such as the BUG$$_{xxx}$ symbols.

All address symbols identify memory locations. SDA generally does not distinguish among different types of address symbols. However, for a symbol identified as the name of a procedure descriptor, SDA takes an additional step of creating an associated symbol to name the code entry point address of the procedure. It forms the code entry point symbol name by appending _C to the name of the procedure descriptor.

Also, SDA substitutes the code entry point symbol name for the procedure descriptor symbol when you enter the following command:

SDA> EXAMINE/INSTRUCTION procedure descriptor
For example, enter the following command:

SDA> EXAMINE/INSTRUCTION SCH$QAST

SDA displays the following information:

SCH$QAST_C: SUBQ SP,#X40,SP

Now enter the EXAMINE command but do not specify the /INSTRUCTION qualifier, as follows:

SDA> EXAMINE SCH$QAST

SDA displays the following information:

SCH$QAST: 0000002C.00003009 "..,.."

This display shows the contents of the first two longwords of the procedure descriptor.

Note that there are no routine address symbols on Alpha systems, except for those in MACRO-64 assembly language modules. Therefore, SDA creates a routine address symbol for every procedure descriptor it has in its symbol table. The new symbol name is the same as for the procedure descriptor except that it has an _C appended to the end of the name.

Sources for SDA Symbols

SDA can get its information from the following places:

- Images (.EXE files)
- Image symbol table files (.STB files)
- Object files

SDA also defines symbols to access registers and to access common data structures.

The only images with symbols are shareable images and executive images. These images contain only universal symbols, such as constants and addresses.

The image symbol table files are produced by the linker with the /SYMBOLS qualifier. These files normally only contain universal symbols, as do the executable images. However, if the SYMBOL_TABLE=GLOBALS linker option is specified, the .STB file also contains all global symbols defined in the image. See the OpenVMS Linker Utility Manual for more information.

Object files can contain global constant values. An object file used with SDA typically contains symbol definitions for data structure fields. Such an object file can be generated by compiling a MACRO-32 source module that invokes specific macros. The macros, which are typically defined in SYS$LIBRARY:LIB.MLB or STARLET.MLB, define symbols that correspond to data structure field offsets. The macro $UCBDEF, for example, defines offsets for fields within a unit control block (UCB). OpenVMS Alpha provides a number of such object modules in SDA$READ_DIR, as listed in Table 2–4. For compatibility with OpenVMS VAX, the modules’ file types have been renamed to .STB.
### Table 2–4 Modules Containing Global Symbols and Data Structures Used by SDA

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCLDEF.STB</td>
<td>Symbols for the DCL interpreter</td>
</tr>
<tr>
<td>DECDTMDEF.STB</td>
<td>Symbols for transaction processing</td>
</tr>
<tr>
<td>GLXDEF.STB</td>
<td>Symbols for OpenVMS Galaxy data structures</td>
</tr>
<tr>
<td>IMGDEF.STB</td>
<td>Symbols for the image activator</td>
</tr>
<tr>
<td>IODEF.STB</td>
<td>I/O database structure symbols</td>
</tr>
<tr>
<td>NETDEF.STB</td>
<td>Symbols for DECnet data structures</td>
</tr>
<tr>
<td>REQSYSDEF.STB</td>
<td>Required symbols for SDA</td>
</tr>
<tr>
<td>RMSDEF.STB</td>
<td>Symbols that define RMS internal and user data structures and RMS$_$xxx completion codes</td>
</tr>
<tr>
<td>SCSDEF.STB</td>
<td>Symbols that define data structures for system communications services</td>
</tr>
<tr>
<td>SYSDEF.STB</td>
<td>Symbols that define system data structures, including the I/O database</td>
</tr>
<tr>
<td>TCPIP$NET_GLOBALS.STB(^1)</td>
<td>Data structure definitions for TCP/IP internet driver, execlet, and ACP data structures</td>
</tr>
<tr>
<td>TCPIP$NFS_GLOBALS.STB(^1)</td>
<td>Data structure definitions for TCP/IP NFS server</td>
</tr>
<tr>
<td>TCPIP$PROXY_GLOBALS.STB(^2)</td>
<td>Data structure definitions for TCP/IP proxy execlet</td>
</tr>
<tr>
<td>TCPIP$PWIP_GLOBALS.STB(^1)</td>
<td>Data structure definitions for TCP/IP PWIP driver, and ACP data structures</td>
</tr>
<tr>
<td>TCPIP$TN_GLOBALS.STB(^1)</td>
<td>Data structure definitions for TCP/IP TELNET/RLOGIN server driver data structures</td>
</tr>
</tbody>
</table>

\(^1\)Only available if TCP/IP has been installed. These are found in SYS$SYSTEM, so that all files are not automatically read in when you issue a READ/EXEC command.

Table 2–5 lists symbols that SDA defines automatically on initialization.

### Table 2–5 SDA Symbols Defined on Initialization

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASN</td>
<td>Address space number</td>
</tr>
<tr>
<td>AST</td>
<td>Both the asynchronous system trap status and enable registers: AST&lt;3:0&gt; = AST enable; AST&lt;7:4&gt; = AST status</td>
</tr>
<tr>
<td>ESP</td>
<td>Executive stack pointer</td>
</tr>
<tr>
<td>FEN</td>
<td>Floating-point enable</td>
</tr>
<tr>
<td>FP</td>
<td>Frame pointer (R29)</td>
</tr>
<tr>
<td>FP0 through FP30</td>
<td>Floating-point registers 0-30</td>
</tr>
<tr>
<td>FPCR</td>
<td>Floating-point control register</td>
</tr>
<tr>
<td>G</td>
<td>$FFFFFF.80000000_{16}$, the base address of system space</td>
</tr>
<tr>
<td>H</td>
<td>$00000000.7FFE0000_{16}$, a base address in P1 space</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 2–5 (Cont.) SDA Symbols Defined on Initialization

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$FFFFFF.FFFFFFFF_{16}$, also fills the leading digits of a hexadecimal number with the value of F</td>
</tr>
<tr>
<td>KSP</td>
<td>Kernel stack pointer</td>
</tr>
<tr>
<td>PC</td>
<td>Program counter</td>
</tr>
<tr>
<td>PCC</td>
<td>Process cycle counter</td>
</tr>
<tr>
<td>PS</td>
<td>Processor status</td>
</tr>
<tr>
<td>PTBR</td>
<td>Page table base register</td>
</tr>
<tr>
<td>R0 through R29</td>
<td>Integer registers</td>
</tr>
<tr>
<td>SCC</td>
<td>System cycle counter</td>
</tr>
<tr>
<td>SP</td>
<td>Current stack pointer of a process</td>
</tr>
<tr>
<td>SSP</td>
<td>Supervisor stack pointer</td>
</tr>
<tr>
<td>USP</td>
<td>User stack pointer</td>
</tr>
</tbody>
</table>

After a SET CPU command is issued (for analyzing a crash dump only), the symbols defined in Table 2–6 are set for that CPU.

Table 2–6 SDA Symbols Defined by SET CPU Command

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUDB</td>
<td>Address of CPU database</td>
</tr>
<tr>
<td>IPL</td>
<td>Interrupt priority level register</td>
</tr>
<tr>
<td>PCBB</td>
<td>Process context block base register</td>
</tr>
<tr>
<td>PRBR</td>
<td>Processor base register (CPU database address)</td>
</tr>
<tr>
<td>SCBB</td>
<td>System control block base register</td>
</tr>
<tr>
<td>SISR</td>
<td>Software interrupt status register</td>
</tr>
</tbody>
</table>

After a SET PROCESS command is issued, the symbols listed in Table 2–7 are defined for that process.

Table 2–7 SDA Symbols Defined by SET PROCESS Command

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARB</td>
<td>Address of access rights block</td>
</tr>
<tr>
<td>JIB</td>
<td>Address of job information block</td>
</tr>
<tr>
<td>KTB</td>
<td>Address of the kernel thread block</td>
</tr>
<tr>
<td>ORB</td>
<td>Address of object rights block</td>
</tr>
<tr>
<td>PCB</td>
<td>Address of process control block</td>
</tr>
<tr>
<td>PHD</td>
<td>Address of process header</td>
</tr>
</tbody>
</table>

Other SDA commands, such as SHOW DEVICE and SHOW CLUSTER, predefined additional symbols.

**SDA Symbol Initialization**

On initialization, SDA reads the universal symbols defined by SYS$BASE_IMAGE.EXE. For every procedure descriptor address symbol found, a routine address symbol is created (with _C appended to the symbol name).
SDA Description

2.6 SDA Command Format

SDA then reads the object file REQSYSDEF.STB. This file contains data structure definitions that are required for SDA to run correctly. It uses these symbols to access some of the data structures in the crash dump file or on the running system.

Finally, SDA initializes the process registers defined in Table 2–7 and executes a SET CPU command, defining the symbols as well.

Use of SDA Symbols

There are two major uses of the address type symbols. First, the EXAMINE command employs them to find the value of a known symbol. For example, EXAMINE CTL$GL_PCB finds the PCB for the current process. Then, certain SDA commands (such as EXAMINE, SHOW STACK, and FORMAT) use them to symbolize addresses when generating output.

When the code for one of these commands needs a symbol for an address, it calls the SDA symbolize routine. The symbolize routine tries to find the symbol in the symbol table whose address is closest to, but not greater than the requested address. This means, for any given address, the routine may return a symbol of the form symbol_name+offset. If, however, the offset is greater than 0FFF16, it fails to find a symbol for the address.

As a last resort, the symbolize routine checks to see if this address falls within a known memory range. Currently, the only known memory ranges are those used by the OpenVMS Alpha executive images and those used by active images in a process. SDA searches through the executive loaded image list (LDRIMG data structure) to see if the address falls within any of the image sections. If SDA does find a match, it returns one of the following types of symbols:

- executive_image_name+offset
- activated_image_name+offset

The offset is the same as the image offset as defined in the map file.

The constants in the SDA symbol table are usually used to display a data structure with the FORMAT command. For example, the PHD offsets are defined in SYSDEF.STB; you can display all the fields of the PHD by entering the following commands:

```
SDA> READ SDA$READ_DIR:SYSDEF.STB
SDA> FORMAT/TYPE=PHD phd_address
```

Symbols and Address Resolution

In OpenVMS Alpha, executive and user images are loaded into dynamically assigned address space. To help you associate a particular virtual address with the image whose code has been loaded at that address, SDA provides several features:

- The SHOW EXECUTIVE command
- The symbolization of addresses, described in the previous section
- The READ command
- The SHOW PROCESS command with the /IMAGES qualifier
- The MAP command
The OpenVMS Alpha executive consists of two base images, SYS$BASE_IMAGE.EXE and SYS$PUBLIC_VECTORS.EXE, and a number of other separately loadable images. Some of these images are loaded on all systems, while others support features unique to particular system configurations. Executive images are mapped into system space during system initialization.

By default, a typical executive image is not mapped at contiguous virtual addresses. Instead, its nonpageable image sections are loaded into a reserved set of pages with other executive images’ nonpageable sections. The pageable sections of a typical executive image are mapped contiguously into a different part of system space. An image mapped in this manner is said to be sliced. A particular system may have system parameters defined that disable executive image slicing altogether.

Each executive image is described by a data structure called a loadable image data block (LDRIMG). The LDRIMG specifies whether the image has been sliced. If the image is sliced, the LDRIMG indicates the beginning of each image section and the size of each section. All the LDRIMGs are linked together in a list that SDA scans to determine what images have been loaded and into what addresses they have been mapped. The SHOW EXECUTIVE command displays a list of all images that are included in the OpenVMS Alpha executive.

Each executive image is a shareable image whose universal symbols are defined in the SYS$BASE_IMAGE.EXE symbol vector. On initialization, SDA reads this symbol vector and adds its universal symbols to the SDA symbol table.

Executive image .STB files define additional symbols within an executive image that are not defined as universal symbols and thus are not in the SYS$BASE_IMAGE.EXE symbol vector (see Sources for SDA Symbols in this section). You can enter a READ/EXECUTIVE command to read symbols defined in all executive image .STB files into the SDA symbol table, or a READ/IMAGE filespec command to read the .STB for a specified image only.

To obtain a display of all images mapped within a process, execute a SHOW PROCESS/IMAGE command. See the description of the SHOW PROCESS command for additional information about displaying the hardware and software context of a process.

You can also identify the image name and offset that correspond to a specified address with the MAP command. With the information obtained from the MAP command, you can then examine the image map to locate the source module and program section offset corresponding to an address.

### 2.7 Investigating System Failures

This section discusses how the operating system handles internal errors, and suggests procedures that can help you determine the causes of these errors. It illustrates, through detailed analysis of a sample system failure, how SDA helps you find the causes of operating system problems.

For a complete description of the commands discussed in the sections that follow, refer to Chapter 4 and Chapter 5 of this document, where all the SDA and CLUE commands are presented in alphabetical order.
2.7 Investigating System Failures

2.7.1 General Procedure for Analyzing System Failures

When the operating system detects an internal error so severe that normal operation cannot continue, it signals a condition known as a fatal bugcheck and shuts itself down. A specific bugcheck code describes each fatal bugcheck.

To resolve the problem, you must find the reason for the bugcheck. Many failures are caused by errors in user-written device drivers or other privileged code not supplied by Compaq. To identify and correct these errors, you need a listing of the code in question.

Occasionally, a system failure is the result of a hardware failure or an error in code supplied by Compaq. A hardware failure requires the attention of Compaq Services. To diagnose an error in code supplied by Compaq, you need listings of that code, which are available from Compaq.

Start the search for the error by analyzing the CLUE list file that was created by default when the system failed. This file contains an overview of the system failure, which can assist you in finding the line of code that signaled the bugcheck. CLUE CRASH displays the content of the program counter (PC) in the list file. The content of the PC is the address of the next instruction after the instruction that signaled the bugcheck.

However, some bugchecks are caused by unexpected exceptions. In such cases, the address of the instruction that caused the exception is more informative than the address of the instruction that signaled the bugcheck. The address of the instruction that caused the exception is located on the stack. You can obtain this address either by using the SHOW STACK command to display the contents of the stack or by using the CLUE CRASH command to display the system state at time of exception. See Section 2.7.2 for information on how to proceed for several types of bugchecks.

Once you have found the address of the instruction that caused the bugcheck or exception, find the module in which the failing instruction resides. Use the MAP command to determine whether the instruction is part of a device driver or another executive image. Alternatively, the SHOW EXECUTIVE command shows the location and size of each of the images that make up the OpenVMS Alpha executive.

If the instruction that caused the bugcheck is not part of a driver or executive image, examine the linker’s map of the module or modules you are debugging to determine whether the instruction that caused the bugcheck is in your program.

To determine the general cause of the system failure, examine the code that signaled the bugcheck or the instruction that caused the exception.

2.7.2 Fatal Bugcheck Conditions

There are many possible conditions that can cause OpenVMS Alpha to issue a bugcheck. Normally, these occasions are rare. When they do occur, they are often fatal exceptions or illegal page faults occurring within privileged code. This section describes the symptoms of several common bugchecks. A discussion of other exceptions and condition handling in general appears in the OpenVMS Programming Concepts Manual.

An exception is fatal when it occurs while either of the following conditions exists:

- The process is executing above IPL 2 (IPL$_ASTDEL).
The process is executing in a privileged (kernel or executive) processor access mode and has not declared a condition handler to deal with the exception.

When the system fails, the operating system reports the approximate cause of the system failure on the console terminal. SDA displays a similar message when you issue a SHOW CRASH command. For instance, for a fatal exception, SDA can display one of these messages:

- **FATALEXCPT**, Fatal executive or kernel mode exception
- **INVEXCEPTN**, Exception while above ASTDEL
- **SSRVEXCEPT**, Unexpected system service exception
- **UNXSIGNAL**, Unexpected signal name in ACP

When a FATALEXCPT, INVEXCEPTN, SSRVEXCEPT, or UNXSIGNAL bugcheck occurs, two argument lists, known as the mechanism and signal arrays, are placed on the stack.

Section 2.7.2.1 to Section 2.7.2.4 describe these arrays and related data structures, and Section 2.7.2.5 shows example output from SDA for an SSRVEXCEPT bugcheck.

A page fault is illegal when it occurs while the interrupt priority level (IPL) is greater than 2 (IPL$_ASTDEL$). When OpenVMS Alpha fails because of an illegal page fault, it displays the following message on the console terminal:

- **PGFIPLHI**, Page fault with IPL too high

Section 2.7.2.6 describes the stack contents when an illegal page fault occurs.

### 2.7.2.1 Mechanism Array

Figure 2–1 illustrates the **mechanism array**, which is made up entirely of quadwords. The first quadword of this array indicates the number of quadwords in this array; this value is always $2^{16}$. These quadwords are used by the procedures that search for a condition handler and report exceptions.
2.7 Investigating System Failures

Figure 2–1  Mechanism Array

<table>
<thead>
<tr>
<th>Mechanism Args</th>
<th>Quadword Aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCH_ARGS</td>
<td></td>
</tr>
<tr>
<td>MCH_FLAGS</td>
<td></td>
</tr>
<tr>
<td>MCH_FRAME</td>
<td></td>
</tr>
<tr>
<td>MCH_DEPTH</td>
<td></td>
</tr>
<tr>
<td>MCH_RESERVED1</td>
<td></td>
</tr>
<tr>
<td>MCH_DADDR</td>
<td></td>
</tr>
<tr>
<td>MCH_ESF_ADDR</td>
<td></td>
</tr>
<tr>
<td>MCH_SIG_ADDR</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR0</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR0_LOW</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR0_HIGH</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR1</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR1_LOW</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR1_HIGH</td>
<td></td>
</tr>
<tr>
<td>MCH_SAVR16</td>
<td></td>
</tr>
</tbody>
</table>

Integer registers 17-27

MCH_SAVR28

MCH_SAVF0

MCH_SAVF1

MCH_SAVF10

Floating registers 11-29

MCH_SAVF30

MCH_SIG64_ADDR

CHF$S_CHFDEF2 = 360
Symbolic offsets into the mechanism array are defined as follows. The SDA SHOW STACK command identifies the elements of the mechanism array on the stack using these symbols.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHF$IS_MCH_ARGS</td>
<td>Number of quadwords that follow. In a mechanism array, this value is always 2C_{16}.</td>
</tr>
<tr>
<td>CHF$IS_MCH_FLAGS</td>
<td>Flag bits for related argument mechanism information.</td>
</tr>
<tr>
<td>CHF$PH_MCH_FRAME</td>
<td>Address of the FP (frame pointer) of the establisher’s call frame.</td>
</tr>
<tr>
<td>CHF$IS_MCH_DEPTH</td>
<td>Depth of the OpenVMS Alpha search for a condition handler.</td>
</tr>
<tr>
<td>CHF$PH_MCH_DADDR</td>
<td>Address of the handler data quadword, if the exception handler data field is present.</td>
</tr>
<tr>
<td>CHF$PH_MCH_ESF_ADDR</td>
<td>Address of the exception stack frame (see Figure 2–4).</td>
</tr>
<tr>
<td>CHF$PH_MCH_SIG_ADDR</td>
<td>Address of the signal array (see Figure 2–2).</td>
</tr>
<tr>
<td>CHF$IH_MCH_SAVRnn</td>
<td>Contents of the saved integer registers at the time of the exception. The following registers are saved: R0, R1, and R16 to R28 inclusive.</td>
</tr>
<tr>
<td>CHF$FH_MCH_SAVFnn</td>
<td>If the process was using floating point, contents of the saved floating-point registers at the time of the exception. The following registers are saved: F0, F1, and F10 to F30 inclusive.</td>
</tr>
<tr>
<td>CHF$PH_MCH_SIG64_ADDR</td>
<td>Address of the 64-bit signal array (see Figure 2–3).</td>
</tr>
</tbody>
</table>

2.7.2.2 Signal Array

The **signal array** appears somewhat farther down the stack. This array comprises all longwords so that the structure is VAX compatible. A signal array describes the exception that occurred. It contains an argument count, the exception code, zero or more exception parameters, the PC, and the PS. Therefore, the size of a signal array can vary from exception to exception. Although there are several possible exception conditions, access violations are most common. Figure 2–2 shows the signal array for an access violation.
For access violations, the signal array is set up as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector list length</td>
<td>Number of longwords that follow. For access violations, this value is always 5.</td>
</tr>
<tr>
<td>Condition value</td>
<td>Exception code. The value 0C₁₆ represents an access violation. You can identify the exception code by using the SDA command EVALUATE/CONDITION_VALUE or SHOW CRASH.</td>
</tr>
<tr>
<td>Additional arguments</td>
<td>These can include a reason mask and a virtual address. In the longword mask if bit 0 of the longword is set, the failing instruction (at the PC saved below) caused a length violation. If bit 1 is set, it referred to a location whose page table entry is in a “no access” page. Bit 2 indicates the type of access used by the failing instruction: it is set for write and modify operations and clear for read operations. The virtual address represents the low-order 32 bits of the virtual address that the failing instruction tried to reference.</td>
</tr>
<tr>
<td>PC</td>
<td>PC whose execution resulted in the exception.</td>
</tr>
<tr>
<td>PS</td>
<td>PS at the time of the exception.</td>
</tr>
</tbody>
</table>
2.7.2.3 64-Bit Signal Array

The **64-bit signal array** also appears further down the stack. This array comprises all quadwords and is not VAX compatible. It contains the same data as the signal array, and Figure 2–3 shows the 64-bit signal array for an access violation. The SDA SHOW STACK command uses the CHF64$ symbols listed in the figure to identify the 64-bit signal array on the stack.

**Figure 2–3 64-Bit Signal Array**

For access violations, the 64-bit signal array is set up as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector list length</td>
<td>Number of quadwords that follow. For access violations, this value is always 5.</td>
</tr>
<tr>
<td>Condition value</td>
<td>Exception code. The value 0C₁₆ represents an access violation. You can identify the exception code by using the SDA command EVALUATE/CONDITION_VALUE or SHOW CRASH.</td>
</tr>
<tr>
<td>Additional arguments</td>
<td>These can include a reason mask and a virtual address.</td>
</tr>
<tr>
<td></td>
<td>In the quadword mask if bit 0 of the quadword is set, the failing instruction (at the PC saved below) caused a length violation. If bit 1 is set, it referred to a location whose page table entry is in a “no access” page. Bit 2 indicates the type of access used by the failing instruction: it is set for write and modify operations and clear for read operations.</td>
</tr>
<tr>
<td>PC</td>
<td>PC whose execution resulted in the exception.</td>
</tr>
<tr>
<td>PS</td>
<td>PS at the time of the exception.</td>
</tr>
</tbody>
</table>
2.7.2.4 Exception Stack Frame

Figure 2–4 illustrates the exception stack frame, which comprises all quadwords.

![Figure 2–4 Exception Stack Frame](image)

The values contained in the exception stack frame are defined as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTSTK$Q_R2</td>
<td>Contents of R2 at the time of the exception</td>
</tr>
<tr>
<td>INTSTK$Q_R3</td>
<td>Contents of R3 at the time of the exception</td>
</tr>
<tr>
<td>INTSTK$Q_R4</td>
<td>Contents of R4 at the time of the exception</td>
</tr>
<tr>
<td>INTSTK$Q_R5</td>
<td>Contents of R5 at the time of the exception</td>
</tr>
<tr>
<td>INTSTK$Q_R6</td>
<td>Contents of R6 at the time of the exception</td>
</tr>
<tr>
<td>INTSTK$Q_R7</td>
<td>Contents of R7 at the time of the exception</td>
</tr>
<tr>
<td>INTSTK$Q_PC</td>
<td>PC whose execution resulted in the exception</td>
</tr>
<tr>
<td>INTSTK$Q_PS</td>
<td>PS at the time of the exception (except high-order bits)</td>
</tr>
</tbody>
</table>

The SDA SHOW STACK command identifies the elements of the exception stack frame on the stack using these symbols.

2.7.2.5 SSRVEXCEPT Example

If OpenVMS Alpha encounters a fatal exception, you can find the code that signaled it by examining the PC in the signal array. Use the SHOW CRASH or CLUE CRASH command to display the PC and the instruction stream around the PC to locate the exception.
The following display shows the SDA output in response to the SHOW CRASH and SHOW STACK commands for an SSRVEXCEPT bugcheck. It illustrates the mechanism array, signal arrays, and the exception stack frame previously described.

OpenVMS (TM) Alpha system dump analyzer
...analyzing a selective memory dump...

Dump taken on 30-AUG-2000 13:13:46.83
SSRVEXCEPT, Unexpected system service exception

SDA> SHOW CRASH

Version of system: OpenVMS (TM) Alpha Operating System, Version V7.3
System Version Major ID/Minor ID: 3/0
System type: DEC 3000 Model 400
Crash CPU ID/Primary CPU ID: 00/00
Bitmask of CPUs active/available: 00000001/00000001

CPU bugcheck codes:

CPU 00 -- SSRVEXCEPT, Unexpected system service exception

System State at Time of Exception
---------------------------------
Exception Frame:
----------------
R2 = 00000000.00000003  R3 = FFFFFFFF.80C63460  EXCEPTION_MON_NPRW+06A60
R4 = FFFFFFFF.80D12740  PCB
R5 = 00000000.000000C8
R6 = 00000000.00030038
R7 = 00000000.7FFA1FC0
PC = 00000000.00030078
PS = 00000000.00000003

00000000.00030068: STQ  R27,(SP)
00000000.0003006C: BIS  R31,SP,FP
00000000.00030070: STQ  R26,#X0010(SP)
00000000.00030074: LDA  R28,(R31)
PC => 00000000.00030078: LDL  R28,(R28)
00000000.0003007C: BEQ  R28,#X000007
00000000.00030080: LDQ  R26,#XFFE8(R27)
00000000.00030084: BIS  R31,R26,R0
00000000.00030088: BIS  R31,FP,SP

PS =>

MBZ SPAL  MBZ  IPL VM& MBZ CURMOD INT PRVMOD
0 00 0000000000 00 0 Kern 0 user

Signal Array
--------------
Length = 00000005
Type = 0000000C
Arg = 00000000.00010000
Arg = 00000000.00000000
Arg = 00000000.00030078
Arg = 00000000.00000003

%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=0000000000000000,
PC=00000000.00030078, PS=00000003
2.7 Investigating System Failures

Saved Scratch Registers in Mechanism Array
------------------------------------------
R0 = 00000000.00020000  R1 = 00000000.00000000  R16 = 00000000.00020004
R17 = 00000000.00010050  R18 = FFFFFFFF.FFFFFFFF  R19 = 00000000.00000000
R20 = 00000000.7FFA1F50  R21 = 00000000.00000000  R22 = 00000000.00010050
R23 = 00000000.00000000  R24 = 00000000.00010051  R25 = 00000000.00000000
R26 = FFFFFFFF.8010ACA4  R27 = 00000000.00010050  R28 = 00000000.00000000

CPU 00 Processor crash information
----------------------------------
CPU 00 reason for Bugcheck: SSRVEXCEPT, Unexpected system service exception

Process currently executing on this CPU: SYSTEM

Current image file: $31$DKB0:[SYS0.][SYSMGR]X.EXE;1

Current IPL: 0 (decimal)

CPU database address: 80D0E000

CPUs Capabilities: PRIMARY,QUORUM,RUN

General registers:
R0 = 00000000.00000000  R1 = 00000000.7FFA1EB8  R2 = FFFFFFFF.80D0E6C0
R3 = FFFFFFFF.80C63460  R4 = FFFFFFFF.80D12740  R5 = 00000000.000000C8
R6 = 00000000.00030038  R7 = 00000000.7FFA1FC0  R8 = 00000000.00000000
R9 = 00000000.7FFAC410  R10 = 00000000.7FFAD238  R11 = 00000000.7FFCE3E0
R12 = 00000000.00000000  R13 = FFFFFFFF.80C6EB60  R14 = 00000000.00000000
R15 = 00000000.009A79FD  R16 = 00000000.000003C4  R17 = 00000000.7FFA1D40
R18 = FFFFFFFF.80C05C38  R19 = 00000000.00000000  R20 = 00000000.7FFA1F50
R21 = 00000000.00000000  R22 = 00000000.00000000  R23 = 00000000.7FFP03C8
R24 = 00000000.7FFP0040  R25 = 00000000.00000003  R26 = 00000000.82A21080
R27 = 00000000.7FFA1CA0  R28 = FFFFFFFF.8004B6DC  FP = 00000000.7FFA1CA0
PC = FFFFFFFF.82A210B4  PS = 18000000.00000000

Processor Internal Registers:
ASN = 00000000.0000002F  ASTSR/ASTEN = 0000000F
IPL = 00000000  PCBB = 00000000.003FE080  PRBR = FFFFFFFF.80D0E000
PTBR = 00000000.00001136  SCBB = 00000000.000001DC  SISR = 00000000.00000000
VPTB = FFFFFFFF.8004B6DC  FPCR = 00000000.00000000  MCES = 00000000.00000000

CPU 00 Processor crash information
----------------------------------
KSP = 00000000.7FFA1C98
ESP = 00000000.7FFA6000
SSP = 00000000.7FFA1C00
USP = 00000000.7FFFA6AD0

No spinlocks currently owned by CPU 00
2.7 Investigating System Failures

SDA Description

SDA> SHOW STACK

Current Operating Stack (KERNEL):

00000000.7FFA1C78 18000000.00000000
00000000.7FFA1C80 00000000.7FFA1CA0
00000000.7FFA1C88 00000000.00000000
00000000.7FFA1C90 00000000.7FFA1D40
SP => 00000000.7FFA1C98 00000000.00000000
00000000.7FFA1C98 00000000.00000000
00000000.7FFA1CC0 FFFFFFFF.829CEDA8 EXE$SET_PAGES_READ_ONLY+00948
00000000.7FFA1CC8 00000000.00000000
00000000.7FFA1CF0 FFFFFFFF.80C63780 EXE$ACVIOLAT
00000000.7FFA1CF8 00000000.7FFA1EB8
00000000.7FFA1D00 00000000.7FFA1D40
00000000.7FFA1D08 00000000.7FFA1F00
00000000.7FFA1D10 00000000.7FFA1F40
00000000.7FFA1D18 00000000.00000000
00000000.7FFA1D20 00000000.00000000
00000000.7FFA1D28 00000000.00020000 SYSSK_VERSION_04
00000000.7FFA1D30 00000000.00000025 BUG$_NETRCVPKT
00000000.7FFA1D38 829CE050.000008F8 BUG$_SEQ_NUM_OVF
CHF$IS_MCHARGS 00000000.7FFA1D40 00000000.0000002C
CHF$PH_MCH_FRAME 00000000.7FFA1D48 00000000.7AFFBADD
CHF$IS_MCHP Tại 00000000.7FFA1D50 FFFFFFFF.FFFFFFD
CHFSH_MCH_DADDR 00000000.7FFA1D58 00000000.00000000
CHF$PH_MCH_ESP_ADDR 00000000.7FFA1D60 00000000.7FFA1F00
CHF$PH_MCH_SIG_ADDR 00000000.7FFA1D68 00000000.7FFA1EB8
CHFSH_MCH_SAVR0 00000000.7FFA1D70 00000000.00020000 SYSSK_VERSION_04
CHFSH_MCH_SAVR1 00000000.7FFA1D78 00000000.00000000
CHFSH_MCH_SAVR16 00000000.7FFA1D80 00000000.00000000
CHFSH_MCH_SAVR17 00000000.7FFA1D88 00000000.00010050 SYSSK_VERSION_16+00010
CHFSH_MCH_SAVR18 00000000.7FFA1D90 FFFFFFFF.FFFFFFD
CHFSH_MCH_SAVR19 00000000.7FFA1D98 00000000.00000000
CHFSH_MCH_SAVR20 00000000.7FFA1DA0 00000000.7FFA1F50
CHFSH_MCH_SAVR21 00000000.7FFA1DA8 00000000.00000000
CHFSH_MCH_SAVR22 00000000.7FFA1DB0 00000000.00010050 SYSSK_VERSION_16+00010
CHFSH_MCH_SAVR23 00000000.7FFA1DB8 00000000.00000000
CHFSH_MCH_SAVR24 00000000.7FFA1DC0 00000000.00010051 SYSSK_VERSION_16+00011
CHFSH_MCH_SAVR25 00000000.7FFA1DC8 00000000.00000000
CHFSH_MCH_SAVR26 00000000.7FFA1DD0 FFFFFFFF.8010ACA4 AMACSEMUL_CALL_NATIVE_C+000A4
CHFSH_MCH_SAVR27 00000000.7FFA1D80 00000000.00010050 SYSSK_VERSION_16+00010
CHFSH_MCH_SAVR28 00000000.7FFA1DE0 00000000.00000000
00000000.7FFA1DB8 00000000.00000000
00000000.7FFA1DF0 00000000.00000000
00000000.7FFA1DF8 00000000.00000000
00000000.7FFA1E00 00000000.00000000
00000000.7FFA1E08 00000000.00000000
00000000.7FFA1E10 00000000.00000000
00000000.7FFA1E18 00000000.00000000
00000000.7FFA1E20 00000000.00000000
00000000.7FFA1E28 00000000.00000000
00000000.7FFA1E30 00000000.00000000
00000000.7FFA1E38 00000000.00000000
00000000.7FFA1E40 00000000.00000000
00000000.7FFA1E48 00000000.00000000
00000000.7FFA1E50 00000000.00000000
00000000.7FFA1E58 00000000.00000000
00000000.7FFA1E60 00000000.00000000
00000000.7FFA1E68 00000000.00000000

SDA Description 2–31
2.7 Investigating System Failures

SDA Description
2.7.2.6 Illegal Page Faults

When an illegal page fault occurs, the stack appears as pictured in Figure 2–5.

**Figure 2–5 Stack Following an Illegal Page-Fault Error**

<table>
<thead>
<tr>
<th>Stack Frame</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMG$PAGEFAULT</td>
<td>Stack frame built at entry to MMG$PAGEFAULT,</td>
</tr>
<tr>
<td></td>
<td>the page fault exception service routine. The</td>
</tr>
<tr>
<td></td>
<td>frame includes the contents of the following</td>
</tr>
<tr>
<td></td>
<td>registers at the time of the page fault: R3,</td>
</tr>
<tr>
<td></td>
<td>R8, R11 to R15, R29 (frame pointer)</td>
</tr>
<tr>
<td>SCH$PAGEFAULT Saved</td>
<td>Contents of the following registers at the time</td>
</tr>
<tr>
<td>Scratch Registers</td>
<td>of the page fault: R0, R1, R16 to R28</td>
</tr>
<tr>
<td>Exception Stack Frame</td>
<td>Exception stack frame (see Figure 2–4)</td>
</tr>
<tr>
<td>Previous Stack Content</td>
<td>Contents of the stack prior to the illegal</td>
</tr>
<tr>
<td></td>
<td>page-fault error</td>
</tr>
</tbody>
</table>

When you analyze a dump caused by a PGFIPLHI bugcheck, the SHOW STACK command identifies the exception stack frame using the symbols shown in Table 2–8. The SHOW CRASH or CLUE CRASH command displays the instruction that caused the page fault and the instructions around it.

2.8 Inducing a System Failure

If the operating system is not performing well and you want to create a dump you can examine, you must induce a system failure. Occasionally, a device driver or other user-written, kernel-mode code can cause the system to execute a loop of code at a high priority, interfering with normal system operation. This loop can occur even though you have set a breakpoint in the code if the loop is encountered before the breakpoint. To gain control of the system in such circumstances, you must cause the system to fail and then reboot it.

If the system has suspended all noticeable activity and is hung, see the examples of causing system failures in Section 2.8.2.

If you are generating a system failure in response to a system hang, be sure to record the PC and PS as well as the contents of the integer registers at the time of the system halt.
2.8 Inducing a System Failure

2.8.1 Meeting Crash Dump Requirements

The following requirements must be met before the operating system can write a complete crash dump:

- You must not halt the system until the console dump messages have been printed in their entirety and the memory contents have been written to the crash dump file. Be sure to allow sufficient time for these events to take place or make sure that all disk activity has stopped before using the console to halt the system.

- There must be a crash dump file in SYS$SPECIFIC:[SYSEXE]: named either SYSDUMP.DMP or PAGEFILE.SYS. This dump file must be either large enough to hold the entire contents of memory (as discussed in Section 2.2.1.1) or, if the DUMPSTYLE system parameter is set, large enough to accommodate a subset or compressed dump (also discussed in Section 2.2.1.1).

If SYSDUMP.DMP is not present, the operating system attempts to write crash dumps to PAGEFILE.SYS. In this case, the SAVEDUMP system parameter must be 1 (the default is 0).

- Alternatively, the system must be set up for DOSD. See Section 2.2.1.5, and the OpenVMS System Manager’s Manual, Volume 2: Tuning, Monitoring, and Complex Systems for details.

- The DUMPBUG system parameter must be 1 (the default is 1).

2.8.2 Procedure for Causing a System Failure

This section tells you how to enter the XDelta utility (XDELTA) to force a system failure.

Before you can use XDelta, it must be loaded at system startup. To load XDelta during system bootstrap, you must set bit 1 in the boot flags. See the OpenVMS Alpha Version 7.1 Upgrade and Installation Manual for information about booting with the XDelta utility.

Put the system in console mode by pressing Ctrl/P or the Halt push button. Enter the following commands at the console prompt to enter XDelta:

```plaintext
>>> DEPOSIT SIRR E
>>> CONTINUE
```

Once you have entered XDelta, use any valid XDelta commands to examine register or memory locations, step through code, or force a system failure (by entering ;C under XDelta). See the OpenVMS Delta/XDelta Debugger Manual for more information about using XDelta.

If you did not load XDelta, you can force a system crash by entering console commands that make the system incur an exception at high IPL. At the console prompt, enter commands to set the program counter (PC) to an invalid address and the PS to kernel mode at IPL 31 before continuing. This results in a forced INVEXCEPTN-type bugcheck. Some Compaq computers employ the console command CRASH (which will force a system failure) while other systems require that you manually enter the commands.

Enter the following commands at the console prompt to force a system failure:

```plaintext
>>> DEPOSIT PC FFFFFFFFFFFFFF00
>>> DEPOSIT PS 1F00
>>> CONTINUE
```
For more information, refer to the hardware manuals that accompanied your computer.
This chapter describes the format, usage, and qualifiers of the System Dump Analyzer (SDA) utility.

### 3.1 ANALYZE Usage Summary

The System Dump Analyzer (SDA) utility helps determine the causes of system failures. This utility is also useful for examining the running system.

**Format**

```
ANALYZE {/CRASH_DUMP [/RELEASE] [/OVERRIDE] filespec | /SYSTEM}
[/SYMBOL = system-symbols-table]
```

**Command Parameter**

- **filespec**
  Name of the file that contains the dump you want to analyze. At least one field of the `filespec` is required, and it can be any field. The default `filespec` is the highest version of SYSDUMP.DMP in your default directory.

**Description**

By default, the System Dump Analyzer is automatically invoked when you reboot the system after a system failure.

To analyze a system dump interactively, invoke SDA by issuing the following command:

```
$ ANALYZE/CRASH_DUMP filespec
```

If you do not specify `filespec`, SDA prompts you for it.

To analyze a crash dump, your process must have the privileges necessary for reading the dump file. This usually requires system privilege (SYSPRV), but your system manager can, if necessary, allow less privileged processes to read the dump files. Your process needs change-mode-to-kernel (CMKRNL) privilege to release page file dump blocks, whether you use the /RELEASE qualifier or the SDA COPY command.

Invoke SDA to analyze a running system by issuing the following command:

```
$ANALYZE/SYSTEM
```

To examine a running system, your process must have change-mode-to-kernel (CMKRNL) privilege. Your process must also have the map-by-PFN privilege (PFNMAP) to access memory by physical address on a running system. You cannot specify `filespec` when using the /SYSTEM qualifier.
ANALYZE Usage Summary and Qualifiers

3.1 ANALYZE Usage Summary

To send all output from SDA to a file, use the SDA command SET OUTPUT, specifying the name of the output file. The file produced is 132 columns wide and is formatted for output to a printer. To later redirect the output to your terminal, use the following command:

SDA> SET OUTPUT SYS$OUTPUT

To send a copy of all the commands you type and a copy of all the output those commands produce to a file, use the SDA command SET LOG, specifying the name of the log file. The file produced is 132 columns wide and is formatted for output to a printer.

To exit from SDA, use the EXIT command. Note that the EXIT command also causes SDA to exit from display mode. Thus, if SDA is in display mode, you must use the EXIT command twice: once to exit from display mode, and a second time to exit from SDA.

3.2 ANALYZE Qualifiers

The following qualifiers described in this section determine whether the object of an SDA session is a crash dump or a running system. They also help create the environment of an SDA session.

/CRASH_DUMP
/OVERRIDE
/RELEASE
/SYMBOL
/SYSTEM
/CRASH_DUMP

Invokes SDA to analyze the specified dump file.

Format

/CRASH_DUMP  filespec

Parameter

filespec
Name of the crash dump file to be analyzed. The default file specification is:

SYS$DISK:[default-dir]SYSDUMP.DMP

SYS$DISK and [default-dir] represent the disk and directory specified in your
last SET DEFAULT command. If you do not specify filespec, SDA prompts you
for it.

Description

See Chapter 2, Section 2.3 for additional information on crash dump analysis.
You cannot specify the /SYSTEM qualifier when you include the /CRASH_DUMP
qualifier in the ANALYZE command.

Examples

1. $ ANALYZE/CRASH_DUMP SYS$SYSTEM:SYSDUMP.DMP
   $ ANALYZE/CRASH SYS$SYSTEM

   These commands invoke SDA to analyze the crash dump stored in
   SYS$SYSTEM:SYSDUMP.DMP.

2. $ ANALYZE/CRASH SYSSYSTEM:PAGEFILE.SYS

   This command invokes SDA to analyze a crash dump stored in the system
   page file.
ANALYZE Usage Summary and Qualifiers

/OVERRIDE

/OVERRIDE

When used with the /CRASH_DUMP qualifier, invokes SDA to analyze only the structure of the specified dump file when a corruption or other problem prevents normal invocation of SDA with the ANALYZE/CRASH_DUMP command.

Format

/CRASH_DUMP/OVERRIDE filespec

Parameter

filespec
Name of the crash dump file to be analyzed. The default file specification is:

SYS$DISK:[default-dir]SYSDUMP.DMP

SYS$DISK and [default-dir] represent the disk and directory specified in your last SET DEFAULT command. If you do not specify filespec, SDA prompts you for it.

Description

See Chapter 2, Section 2.3 for additional information on crash dump analysis. Note that when SDA is invoked with /OVERRIDE, not all the commands in Chapter 2, Section 2.3 can be used. Commands that can be used are as follows:

- Output control commands such as SET OUTPUT and SET LOG
- Dump file related commands such as SHOW DUMP and CLUE ERRLOG

Commands that cannot be used are as follows:

- Commands that access memory addresses within the dump file such as EXAMINE and SHOW SUMMARY
- The /RELEASE qualifier when you include the /OVERRIDE qualifier in the ANALYZE/CRASH_DUMP command

When /OVERRIDE is used, the SDA command prompt is SDA>>.

Examples

1. $ ANALYZE/CRASH_DUMP/OVERRIDE SYS$SYSTEM:SYSDUMP.DMP
   $ ANALYZE/CRASH/OVERRIDE SYS$SYSTEM

   These commands invoke SDA to analyze the crash dump stored in SYS$SYSTEM:SYSDUMP.DMP.
/RELEASE

Invokes SDA to release those blocks in the specified system page file occupied by a crash dump.

Requires CMKRNL (change-mode-to-kernel) privilege.

Format

/RELEASE filespec

Parameter

filespec

Name of the system page file (SYS$SYSTEM:PAGEFILE.SYS). Because the default file specification is SYS$DISK:[default-dir]SYSDUMP.DMP, you must identify the page file explicitly. SYS$DISK and [default-dir] represent the disk and directory specified in your last DCL command SET DEFAULT. If you do not specify filespec, SDA prompts you for it.

Description

Use the /RELEASE qualifier to release from the system page file those blocks occupied by a crash dump. When invoked with the /RELEASE qualifier, SDA immediately deletes the dump from the page file and allows no opportunity to analyze its contents.

When you specify the /RELEASE qualifier in the ANALYZE command, do the following:

1. Use the /CRASH_DUMP qualifier.
2. Include the name of the system page file (SYS$SYSTEM:PAGEFILE.SYS) as the filespec.

If you do not specify the system page file or the specified page file does not contain a dump, SDA generates the following messages:

%SDA-E-BLKSNRSLD, no dump blocks in page file to release, or not page file
%SDA-E-NOTPAGFIL, specified file is not the page file

You cannot specify the /OVERRIDE qualifier when you include the /RELEASE qualifier in the ANALYZE/CRASH_DUMP command.

Example

$ ANALYZE/CRASH_DUMP/RELEASE SYS$SYSTEM:PAGEFILE.SYS
$ ANALYZE/CRASH/RELEASE PAGEFILE.SYS

These commands invoke SDA to release to the page file those blocks in SYS$SYSTEM:PAGEFILE.SYS occupied by a crash dump.
ANALYZE Usage Summary and Qualifiers

/SYMBOL

/SYMBOL

Specifies an alternate system symbol table for SDA to use.

Format

/SYMBOL = system-symbol-table

File specification of the OpenVMS Alpha SDA system symbol table required by SDA to analyze a system dump or running system. The specified system-symbol-table must contain those symbols required by SDA to find certain locations in the executive image.

If you do not specify the /SYMBOL qualifier, SDA uses SDA$READ_DIR:SYS$BASE_IMAGE.EXE to load system symbols into the SDA symbol table. When you specify the /SYMBOL qualifier, SDA assumes the default disk and directory to be SYS$DISK:[ ], that is, the disk and directory specified in your last DCL command SET DEFAULT. If you specify a file for this parameter that is not a system symbol table, SDA exits with a fatal error.

Description

The /SYMBOL qualifier allows you to specify a system symbol table to load into the SDA symbol table. You can use the /SYMBOL qualifier whether you are analyzing a system dump or a running system. It is not normally necessary to use the /SYMBOL qualifier when analyzing the running system, since the default SYS$BASE_IMAGE.EXE is the one in use in the system. However if SDA$READ_DIR has been redefined during crash dump analysis, then the /SYMBOL qualifier can be used to ensure that the correct base image is found when analyzing the running system.

The /SYMBOL qualifier can be used with the /CRASH_DUMP and /SYSTEM qualifiers. It is ignored when /OVERRIDE or /RELEASE is specified.

Example

$ ANALYZE/CRASH_DUMP/SYMBOL=SDA$READ_DIR:SYS$BASE_IMAGE.EXE SYS$SYSTEM

This command invokes SDA to analyze the crash dump stored in SYS$SYSTEM:SYSDUMP.DMP, using the base image in SDA$READ_DIR.
/SYSTEM

Invokes SDA to analyze a running system.
Requires CMKRNL (change-mode-to-kernel) privilege. Also requires PFNMAP (map-by-PFN) privilege to access memory by physical address.

Format

/SYSTEM

Parameters

None.

Description

See Chapter 2, Section 2.4 to use SDA to analyze a running system.
You cannot specify the /CRASH_DUMP, /OVERRIDE, or /RELEASE qualifiers when you include the /SYSTEM qualifier in the ANALYZE command.

Example

$ ANALYZE/SYSTEM

This command invokes SDA to analyze the running system.
This chapter describes the SDA commands that can be used to analyze a system dump or a running system. SDA CLUE extension commands, which can summarize information provided by certain SDA commands and provide additional detail for some SDA commands, are described in Chapter 5.

The SDA commands are as follows:

@ (Execute Command)
ATTACH
COPY
DEFINE
DEFINE/KEY
DUMP
EVALUATE
EXAMINE
EXIT
FORMAT
HELP
MAP
MODIFY DUMP
READ
REPEAT
SEARCH
SET CPU
SET ERASE_SCREEN
SET FETCH
SET LOG
SET OUTPUT
SET PROCESS
SET RMS
SET SIGN_EXTEND
SET SYMBOLIZE
SHOW ADDRESS
SHOW BUGCHECK
SHOW CALL_FRAME
SHOW CLUSTER
SHOW CONNECTIONS
SHOW CPU
SHOW CRASH
SHOW DEVICE
SHOW DUMP
SHOW EXECUTIVE
SHOW GALAXY
SHOW GCT
SHOW GLOBAL SECTION TABLE
SDA Commands

- SHOW GLOCK
- SHOW GMDB
- SHOW GSD
- SHOW HEADER
- SHOW LAN
- SHOW LOCKS
- SHOW MACHINE_CHECK
- SHOW MEMORY
- SHOW PAGE_TABLE
- SHOW PARAMETER
- SHOW PFN_DATA
- SHOW POOL
- SHOW PORTS
- SHOW PROCESS
- SHOW RAD
- SHOW RESOURCES
- SHOW RMD
- SHOW RMS
- SHOW RSPID
- SHOW SHM_CPP
- SHOW SHM_REG
- SHOW SPINLOCKS
- SHOW STACK
- SHOW SUMMARY
- SHOW SYMBOL
- SHOW TQE
- SHOW WORKING_SET_LIST
- SPAWN
- UNDEFINE
- VALIDATE PFN_LIST
- VALIDATE QUEUE
- VALIDATE SHM_CPP
@(Execute Command)

Causes SDA to execute SDA commands contained in a file. Use this command to execute a set of frequently used SDA commands.

Format

@filespec

Parameter

filespec
Name of a file that contains the SDA commands to be executed. The default file type is .COM.

Example

SDA> @USUAL

The Execute command executes the following commands, as contained in a file named USUAL.COM:

    SET OUTPUT LASTCRASH.LIS
    SHOW CRASH
    SHOW PROCESS
    SHOW STACK
    SHOW SUMMARY

This command procedure first makes the file LASTCRASH.LIS the destination for output generated by subsequent SDA commands. Next, the command procedure sends information to the file about the system failure and its context, including a description of the process executing at the time of the failure, the contents of the stack on which the failure occurred, and a list of the processes active on the system.

An EXIT command within a command procedure terminates the procedure at that point, as would an end-of-file.

Command procedures cannot be nested.
ATTACH

Switches control of your terminal from your current process to another process in your job (for example, one created with the SDA SPAWN command).

Format

ATTACH [/PARENT] process-name

Parameter

process-name
Name of the process to which you want to transfer control.

Qualifier

/PARENT
Transfers control of the terminal to the current process parent process. When you specify this qualifier, you cannot specify the process-name parameter.

Examples

1. SDA> ATTACH/PARENT
   This ATTACH command attaches the terminal to the parent process of the current process.

2. SDA> ATTACH DUMPER
   This ATTACH command attaches the terminal to a process named DUMPER in the same job as the current process.
COPY

Copies the contents of the dump file to another file.

Format

COPY [/qualifier...] output-filespec

Parameter

output-filespec
Name of the device, directory, and file to which SDA copies the dump file. The default file specification is:

SYS$DISK:[default-dir]filename.DMP

SYS$DISK and [default-dir] represent the disk and directory specified in your last DCL command SET DEFAULT. You must specify a file name.

Qualifiers

/COMPRESS
Causes SDA to compress dump data as it is writing a copy. If the dump being analyzed is already compressed, then SDA does a direct COPY, and issues an informational message indicating that it is ignoring the /COMPRESS qualifier.

/DECOMPRESS
Causes SDA to decompress dump data as it is writing a copy. If the dump being analyzed is already decompressed, then SDA does a direct COPY, and issues an informational message indicating that it is ignoring the /DECOMPRESS qualifier.

Description

Each time the system fails, the contents of memory and the hardware context of the current process (as directed by the DUMPSTYLE parameter) are copied into the file SYS$SYSTEM:SYSDUMP.DMP (or the page file), overwriting its contents. If you do not save this crash dump elsewhere, it will be overwritten the next time that the system fails.

The COPY command allows you to preserve a crash dump by copying its contents to another file. It is generally useful to invoke SDA during system initialization to execute the COPY command. This ensures that a copy of the dump file is made only after the system has failed. The preferred method for doing this, using the logical name CLUE$SITE_PROC, is described in Section 2.2.3.

The COPY command does not affect the contents of the file containing the dump being analyzed.

If you are using the page file (SYS$SYSTEM:PAGEFILE.SYS) as the dump file instead of SYSDUMP.DMP, successful completion of the COPY command will automatically cause the blocks of the page file containing the dump to be released, thus making them available for paging. Even if the copy operation succeeds, the release operation requires that your process have change-mode-to-kernel (CMKRNL) privilege. Once the dump pages have been released from the page file, the dump information in these pages will be lost and SDA will
**SDA Commands**

**COPY**

Immediately exit. You must perform subsequent analysis upon the copy of the dump created by the COPY command.

If you press Ctrl/T while using the COPY command, the system displays how much of the file has been copied.

**Example**

```
SDA> COPY SYSSCRASH:SAVEDUMP
```

The COPY command copies the dump file into the file SYSSCRASH:SAVEDUMP.DMP.
DEFINE

Assigns a value to a symbol.

Format

DEFINE  [/qualifier...] symbol-name [=] expression

Parameters

symbol-name
Name, containing from 1 to 31 alphanumeric characters, that identifies the symbol. See Chapter 2, Section 2.6.2.4 for a description of SDA symbol syntax and a list of default symbols.

expression
Definition of the symbol’s value. See Chapter 2, Section 2.6.2 for a discussion of the components of SDA expressions.

Qualifier

/PD
Defines a symbol as a procedure descriptor (PD). It also defines the routine address symbol corresponding to the defined symbol (the routine address symbol has the same name as the defined symbol, only with _C appended to the symbol name). See Section 2.6.2.4 for more information about symbols.

Description

The DEFINE command causes SDA to evaluate an expression and then assign its value to a symbol. Both the DEFINE and EVALUATE commands perform computations to evaluate expressions. DEFINE adds symbols to the SDA symbol table but does not display the results of the computation. EVALUATE displays the result of the computation but does not add symbols to the SDA symbol table.

Examples

1. SDA> DEFINE BEGIN = 80058E00
   SDA> DEFINE END = 80058E60
   SDA> EXAMINE BEGIN:END

   In this example, DEFINE defines two addresses, called BEGIN and END. These symbols serve as reference points in memory, defining a range of memory locations for the EXAMINE command to inspect.

2. SDA> DEFINE NEXT = @PC
   SDA> EXAMINE/INSTRUCTION NEXT
   NEXT:  HALT

   The symbol NEXT defines the address contained in the program counter, so that the symbol can be used in an EXAMINE/INSTRUCTION command.
3. SDA> DEFINE VEC SCH$GL_PCBVEC
SDA> EXAMINE VEC
SCH$GL_PCBVEC: 00000000.8060F2CC "iö'....."
SDA>

After the value of global symbol SCH$GL_PCBVEC has been assigned to
the symbol VEC, the symbol VEC is used to examine the memory location or
value represented by the global symbol.

4. SDA> DEFINE/PD VEC SCH$QAST
SDA> EXAMINE VEC
SCH$QAST: 000002C.00003008 ".0...,...."
SDA> EXAMINE VEC_C
SCH$QAST_C: B75E0008.43C8153E ">.ÉC..^.*"
SDA>

In this example, the DEFINE/PD command defines not only the symbol VEC,
but also the corresponding routine address symbol (VEC_C).
**DEFINE/KEY**

Associates an SDA command with a terminal key.

**Format**

```
DEFINE/KEY [/qualifier...] key-name command
```

**Parameters**

- **key-name**
  
  Name of the key to be defined. You can define the following keys under SDA:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Key Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF1</td>
<td>LK201, VT100</td>
</tr>
<tr>
<td>PF2</td>
<td>LK201, VT100</td>
</tr>
<tr>
<td>PF3</td>
<td>LK201, VT100</td>
</tr>
<tr>
<td>PF4</td>
<td>LK201, VT100</td>
</tr>
<tr>
<td>KP0 . . . KP9</td>
<td>Keypad 0–9</td>
</tr>
<tr>
<td>PERIOD</td>
<td>Keypad period</td>
</tr>
<tr>
<td>COMMA</td>
<td>Keypad comma</td>
</tr>
<tr>
<td>MINUS</td>
<td>Keypad minus</td>
</tr>
<tr>
<td>ENTER</td>
<td>Keypad ENTER</td>
</tr>
<tr>
<td>UP</td>
<td>Up arrow</td>
</tr>
<tr>
<td>DOWN</td>
<td>Down arrow</td>
</tr>
<tr>
<td>LEFT</td>
<td>Left arrow</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Right arrow</td>
</tr>
<tr>
<td>E1</td>
<td>LK201 Find</td>
</tr>
<tr>
<td>E2</td>
<td>LK201 Insert Here</td>
</tr>
<tr>
<td>E3</td>
<td>LK201 Remove</td>
</tr>
<tr>
<td>E4</td>
<td>LK201 Select</td>
</tr>
<tr>
<td>E5</td>
<td>LK201 Prev Screen</td>
</tr>
<tr>
<td>E6</td>
<td>LK201 Next Screen</td>
</tr>
<tr>
<td>HELP</td>
<td>LK201 Help</td>
</tr>
<tr>
<td>DO</td>
<td>LK201 Do</td>
</tr>
<tr>
<td>F7 . . . F20</td>
<td>LK201 Function keys</td>
</tr>
</tbody>
</table>

- **command**
  
  SDA command to define a key. The command must be enclosed in quotation marks (" ").

**Qualifiers**

- **/IF_STATE=state_list**
  
  Specifies a list of one or more states, one of which must be in effect for the key definition to work. The /NOIF_STATE qualifier has the same meaning as /IF_STATE=current_state. The state name is an alphanumeric string. States are...
established with the /SET_STATE qualifier. If you specify only one state name, you can omit the parentheses. By including several state names, you can define a key to have the same function in all the specified states.

/KEY
Defines a key as an SDA command. To issue the command, press the defined key and the Return key. If you use the /TERMINATE qualifier as well, you do not have to press the Return key. The /KEY qualifier must be specified.

/LOCK_STATE
/NOLOCK_STATE
Specifies that the state set by the /SET_STATE qualifier remains in effect until explicitly changed. By default, the /SET_STATE qualifier is in effect only for the next definable key you press or the next read-terminating character that you type. This qualifier can be specified only with the /SET_STATE qualifier.

/NOLOCK_STATE is the default.

/SET_STATE=state-name
/NOSET_STATE
Causes the key being defined to create a key state change instead of or in addition to issuing an SDA command. When you use the /SET_STATE qualifier, you supply the name of a key state to be used with the /IF_STATE qualifier in other key definitions.

For example, you can define the PF1 key as the GOLD key and use the /IF_STATE=GOLD qualifier to allow two definitions for the other keys, one in the GOLD state and one in the non-GOLD state. For more information on using the /IF_STATE qualifier, see the DEFINE/KEY command in the OpenVMS DCL Dictionary: A–M.

/NOSET_STATE is the default.

/TERMINATE
/NOTERMINATE
Causes the key definition to include termination of the command, which causes SDA to execute the command when the defined key is pressed. Therefore, you do not have to press the Return key after you press the defined key if you specify the /TERMINATE qualifier.

Description
The DEFINE/KEY command causes an SDA command to be associated with the specified key, in accordance with any of the specified qualifiers described previously.

If the symbol or key is already defined, SDA replaces the old definition with the new one. Symbols and keys remain defined until you exit from SDA.

Examples
1. SDA> DEFINE/KEY PF1 "SHOW STACK"
   SDA> PF1 SHOW STACK [RETURN]
   Process stacks (on CPU 00)
   ---------------------------
   Current operating stack (KERNEL):

The DEFINE/KEY command defines PF1 as the SHOW STACK command. When you press the PF1 key, SDA displays the command and waits for you to
press the Return key.

2. SDA> DEFINE/KEY/TERMINATE PF1 "SHOW STACK"
   SDA> PF1 SHOW STACK
   Process stacks (on CPU 00)
   -----------------------------
   Current operating stack (KERNEL):
   00000000.7FF95D00 00000000.0000000B
   00000000.7FF95D08 FFFFFFFF.804395C8 MMG$TBI_DATA_64+000B8
   00000000.7FF95D10 00000000.00000000
   00000000.7FF95D18 0000FE00.00007E04
   SP => 00000000.7FF95D20 00000000.00000800 IRPSM_EXTEND
   00000000.7FF95D28 00000001.000002F7 UCB$B_PI_FKB+0000B
   00000000.7FF95D30 FFFFFFFF.804395C8 MMG$TBI_DATA_64+000B8
   00000000.7FF95D38 00000002.00000000
   .
   .

   The DEFINE/KEY command defines PF1 as the SDA SHOW STACK command. The /TERMINATE qualifier causes SDA to execute the SHOW STACK command without waiting for you to press the Return key.

3. SDA> DEFINE/KEY/SET_STATE="GREEN" PF1 ""
   SDA> DEFINE/KEY/TERMINATE/IF_STATE=GREEN PF3 "SHOW STACK"
   SDA> PF1 PF3 SHOW STACK
   Process stacks (on CPU 00)
   -----------------------------
   Current operating stack (KERNEL):
   .
   .
   .

   The first DEFINE/KEY command defines PF1 as a key that sets a command state GREEN. The trailing pair of quotation marks is required syntax, indicating that no command is to be executed when this key is pressed.

   The second DEFINE command defines PF3 as the SHOW STACK command, but using the /IF_STATE qualifier makes the definition valid only when the command state is GREEN. Thus, you must press PF1 before pressing PF3 to issue the SHOW STACK command. The /TERMINATE qualifier causes the command to execute as soon as you press the PF3 key.
SDA Commands

DUMP

Displays the contents of a range of memory formatted as a comma-separated variable (CSV) list, suitable for inclusion in a spreadsheet.

Format

DUMP range

[ /LONGWORD (default) | /QUADWORD ]
[ /DECIMAL | /HEXADECIMAL (default) ]
[ /FORWARD (default) | /REVERSE ]
[ /RECORD_SIZE=size ] (default = 512)
[ /INDEX_ARRAY [= { LONGWORD (default) | QUADWORD } ] ]
[ /INITIAL_POSITION = { ADDRESS = address | RECORD = number } ]
[ /COUNT = { ALL | records } ] (default = all records)
[ /PHYSICAL ]

Parameter

range
The range of locations to be displayed. The range is specified in one of the following formats:

m:n   Range from address m to address n inclusive
m;n   Range from address m for n bytes

Qualifiers

/COUNT= { ALL | records }
Gives the number of records to be displayed. The default is to display all records.

/DECIMAL
Outputs data as decimal values.

/FORWARD
Causes SDA to display the records in the history buffer in ascending address order. This is the default.

/HEXADECIMAL
Outputs data as hexadecimal values. This is the default.

/INDEX_ARRAY [= { LONGWORD (default) | QUADWORD } ]
Indicates to SDA that the range of addresses given is a vector of pointers to the records to be displayed. The vector can be a list of longwords (default) or quadwords. The size of the range must be an exact number of longwords or quadwords as appropriate.

/INITIAL_POSITION = { ADDRESS = address | RECORD = number }
Indicates to SDA which record is to be displayed first. The default is the lowest addressed record if /FORWARD is used, and the highest addressed record if /REVERSE is used. The initial position may be given as a record number within the range, or the address at which the record is located.

/LONGWORD
Outputs each data item as a longword. This is the default.
**SDA Commands**

**DUMP**

/PHYSICAL
Indicates to SDA that all addresses (range and/or start position) are physical addresses. By default, virtual addresses are assumed.

/QUADWORD
Outputs each data item as a quadword.

/RECORD_SIZE=size
Indicates the size of each record within the history buffer, the default being 512 bytes. Note that this size must exactly divide into the total size of the address range to be displayed, unless /INDEX_ARRAY is specified.

/REVERSE
Causes SDA to display the records in the history buffer in descending address order.

**Description**

The DUMP command displays the contents of a range of memory formatted as a comma-separated variable (CSV) list, suitable for inclusion in a spreadsheet. It is intended for use with a "history" buffer containing records of information of which the most recently written entry is in the middle of the memory range.

**Note**

See SET OUTPUT/NOHEADER for related information.

**Examples**

1. SDA> DUMP dump g;200/initial_position=record=5/record_size=20/reverse
   05,A77B0010,A79B0008,6B9C4001,47FF041F,A03E0000,47DF041C,201F0016,083 04,A03E0000,47DF041C,201F0058,083,A77B0010,A79B0008,6B9C4001,47FF041F 03,A03E0000,47DF041C,201F0075,083,A03E0000,47DF041C,201F001B,083 02,A77B0010,A79B0008,6B9C4001,47FF041F,A03E0000,47DF041C,201F0074,083 01,43E05120,083,6BFA8001,47FF041F,A77B0010,A79B0008,6B9C4001,47FF041F 0,201F0104,6BFA8001,47FF041F,47FF041F,201F0001,6BFA8001,47FF041F,47FF041F 0F,A03E0000,47DF041C,201F0065,083,A03E0000,47DF041C,201F0006,083 0E,A03E0000,47DF041C,201F001C,083,A03E0000,47DF041C,201F001A,083 0D,A03E0000,47DF041C,201F0071,083,A03E0000,47DF041C,201F0057,083 0C,A03E0000,47DF041C,201F002B,083,A03E0000,47DF041C,201F003A,083 0B,A03E0000,47DF041C,201F007D,083,A77B0010,A79B0008,6B9C4001,47FF041F 0A,A03E0000,47DF041C,201F005A,083,A03E0000,47DF041C,201F0078,083 09,A03E0000,47DF041C,201F0002,082,A03E0000,47DF041C,201F0037,083 08,A03E0000,47DF041C,201F0035,083,A03E0000,47DF041C,201F007A,083 07,A03E0000,47DF041C,201F0019,083,A03E0000,47DF041C,201F0034,083 06,A77B0010,A79B0008,6B9C4001,47FF041F,A03E0000,47DF041C,201F0018,083

This example shows the dump of an area of memory treated as 16 records of 32 bytes each, beginning at record 5, and dumped in reverse order. Note the record number in the first field, and that the dump wraps to the end of the memory area once the first record has been output.
2. SDA> examine SMP$GL_CPU_DATA;80
   00000000 00000000 8FE26000 8FE14000 00000000 00000000 8FE02000 811FE000 ...
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 ...
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 ...
   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 ...

SDA> dump SMP$GL_CPU_DATA;80/index_array/record_size=20/count=5
   0,810C17C0,8EC7C180,026A09C0,02,0,FFFFFFFF,0,0
   01,810C17C0,8EC7C400,026A09C0,02,0,FFFFFFFF,0,01
   04,810C17C0,8EC7CB80,026A09C0,02,0,FFFFFFFF,0,04

This example shows the contents of the CPU database vector, then dumps the first 32 bytes of each CPU database entry. Note that only the first five entries in the array are requested, and those containing zero are ignored.
**EVALUATE**

Computes and displays the value of the specified expression in both hexadecimal and decimal. Alternative evaluations of the expression are available with the use of the qualifiers defined for this command.

**Format**

EVALUATE  [{/CONDITION_VALUE | /PS | /PTE | /SYMBOLS | /TIME}] expression

**Parameter**

expression

SDA expression to be evaluated. Chapter 2, Section 2.6.2 describes the components of SDA expressions.

**Qualifiers**

/CONDITION_VALUE

Displays the message that the $GETMSG system service obtains for the value of the expression.

/PS

Evaluates the specified expression in the format of a processor status.

/PTE

Interprets and displays the expression as a page table entry (PTE). The individual fields of the PTE are separated and an overall description of the PTE's type is provided.

/SYMBOLS

Specifies that all symbols known to be equal to the evaluated expression are to be listed in alphabetical order. The default behavior of the EVALUATE command displays only the first five symbols.

/TIME

Interprets and displays the expression as a 64-bit time value. Positive values are interpreted as absolute time; negative values are interpreted as delta time.

**Description**

If you do not specify a qualifier, the EVALUATE command interprets and displays the expression as hexadecimal and decimal values. In addition, if the expression is equal to the value of a symbol in the SDA symbol table, that symbol is displayed. If no symbol with this value is known, the next lower valued symbol is displayed with an appropriate offset unless the offset is extremely large. (See Section 2.6.2.4 for a description of how SDA displays symbols and offsets.) The DEFINE command adds symbols to the SDA symbol table but does not display the results of the computation. EVALUATE displays the result of the computation but does not add symbols to the SDA symbol table.
Examples

1. SDA> EVALUATE -1
   Hex = FFFFFFFF.FFFFFFFF  Decimal = -1

   The EVALUATE command evaluates a numeric expression, displays the
   value of that expression in hexadecimal and decimal notation, and displays a
   symbol that has been defined to have an equivalent value.

2. SDA> EVALUATE 1
   Hex = 00000000.00000001  Decimal = 1
   CHF$M_CALEXT_CANCEL
   CHF$M_FPREGS_VALID
   CHF$V_CALEXT_LAST
   IRP$M_BUFIO
   IRP$M_CLN_READY
   (remaining symbols suppressed by default)

   The EVALUATE command evaluates a numeric expression and displays the
   value of that expression in hexadecimal and decimal notation. This example
   also shows the symbols that have the displayed value. A maximum of five
   symbols are displayed by default.

3. SDA> DEFINE TEN = A
   SDA> EVALUATE TEN
   Hex = 00000000.0000000A  Decimal = 10
   IRP$B_TYPE
   IRP$S_FMOD
   IRP$V_MBXIO
   TEN
   UCB$B_TYPE
   (remaining symbols suppressed by default)

   This example shows the definition of a symbol named TEN. The EVALUATE
   command then shows the value of the symbol.

   Note that A, the value assigned to the symbol by the DEFINE command,
   could be a symbol. When SDA evaluates a string that can be either a symbol
   or a hexadecimal numeral, it first searches its symbol table for a definition
   of the symbol. If SDA finds no definition for the string, it evaluates the string
   as a hexadecimal number.

4. SDA> EVALUATE (((TEN * 6) + (-1/4)) + 6)
   Hex = 00000000.00000042  Decimal = 66

   This example shows how SDA evaluates an expression of several terms,
   including symbols and rational fractions. SDA evaluates the symbol,
   substitutes its value in the expression, and then evaluates the expression.
   Note that the fraction -1/4 is truncated to 0.

5. SDA> EVALUATE/CONDITION 80000018
   %SYSTEM-W-EXQUOTA, exceeded quota

   This example shows the output of an EVALUATE/CONDITION command.
6. **SDA> EVALUATE/PS 0B03**
   
   SDA interprets the entered value 0B03 as though it were a processor status (PS) and displays the resulting field values.

7. **SDA> EVALUATE/PTE 0BCDFFEE**

   The **EVALUATE/PTE** command displays the expression 0BCDFFEE as a page table entry (PTE) and labels the fields. It also describes the status of the page.

8. **SDA> EVALUATE/TIME 009A9A4C.843DBA9F**
   
   This example shows the use of the **EVALUATE/TIME** command.
EXAMINE

Displays either the contents of a location or range of locations in physical memory, or the contents of a register. Use location parameters to display specific locations or use qualifiers to display the entire process and system regions of memory.

Format

EXAMINE [/qualifier[,...]] [location]

Parameter

location
Location in memory to be examined. A location can be represented by any valid SDA expression. (See Chapter 2, Section 2.6.2 for additional information about expressions.) To examine a range of locations, use the following syntax:

m:n Range of locations to be examined, from m to n
m;n Range of locations to be examined, starting at m and continuing for n bytes

The default location that SDA uses is initially 0 in the program region (P0) of the process that was executing at the time the system failed (if you are examining a crash dump) or your process (if you are examining the running system). Subsequent uses of the EXAMINE command with no parameter specified increase the last address examined by eight. Use of the /INSTRUCTION qualifier increases the default address by four. To examine memory locations of other processes, you must use the SET PROCESS command.

Qualifiers

/ALL
Examines all the locations in the program, and control regions and system space, displaying the contents of memory in hexadecimal longwords and ASCII characters. Do not specify parameters when you use this qualifier.

/CONDITION_VALUE
Examines the specified longword, displaying the message that the $GETMSG system service obtains for the value in the longword.

/INSTRUCTION
Translates the specified range of memory locations into assembly instruction format. Each symbol in the EXAMINE expression that is defined as a procedure descriptor is replaced with the code entry point address of that procedure, unless you also specify the /NOPD qualifier.

/NOPD
Can be used with the /INSTRUCTION qualifier to override treating symbols as procedure descriptors. You can place the qualifier immediately after the /INSTRUCTION qualifier, or following a symbol name.

For more details on using the /NOPD qualifier, see the description for the /PD qualifier.
/NOSUPPRESS
Inhibits the suppression of zeros when displaying memory with one of the following qualifiers: /ALL, /P0, /P1, /SYSTEM, or when a range is specified.

/P0
Displays the entire program region for the default process. Do not specify parameters when you use this qualifier.

/P1
Displays the entire control region for the default process. Do not specify parameters when you use this qualifier.

/PD
Causes the EXAMINE command to treat the location specified in the EXAMINE command as a procedure descriptor (PD). PD can also be used to qualify symbols. The /PD and /NOPD qualifiers can be used with the /INSTRUCTION qualifier to override treating symbols as procedure descriptors. Placing the qualifier right after a symbol will override how the symbol is treated. /PD will force it to be a procedure descriptor, and /NOPD will force it to not be a procedure descriptor. Only the /PD qualifier can be placed right after the /INSTRUCTION qualifier. It treats the calculated value as a process descriptor.

In the following examples, TEST_ROUTINE is a PD symbol. Its value is 500 and the code address in this procedure descriptor is 1000. The first example displays instructions starting at 520.

```
EXAMINE/INSTRUCTION TEST_ROUTINE/NOPD+20
```
The next example fetches code address from TEST_ROUTINE PD, adds 20 and displays instructions at that address. In other words, it displays code starting at location 1020.

```
EXAMINE/INSTRUCTION TEST_ROUTINE+20
```
The final example treats the address TEST_ROUTINE+20 as a procedure descriptor, so it fetches the code address out of a procedure descriptor at address 520. It then uses that address to display instructions.

```
EXAMINE/INSTRUCTION/PD TEST_ROUTINE/NOPD+20
```

/PHYSICAL
Examines physical addresses. The /PHYSICAL qualifier cannot be used in combination with the /P0, /P1, or /SYSTEM qualifiers.

/PS
Examines the specified quadword, displaying its contents in the format of a processor status. This qualifier must precede any parameters used in the command line.

/PTE
Interprets and displays the specified quadword as a page table entry (PTE). The display separates individual fields of the PTE and provides an overall description of the PTE’s type.

/SYSTEM
Displays portions of the writable system region. Do not specify parameters when you use this qualifier.
SDA Commands

EXAMINE

/TIME
Examines the specified quadword, displaying its contents in the format of a system-date-and-time quadword.

Description
The following sections describe how to use the EXAMINE command.

Examining Locations
When you use the EXAMINE command to look at a location, SDA displays the location in symbolic notation (symbolic name plus offset), if possible, and its contents in hexadecimal and ASCII formats:

SDA> EXAMINE G6605C0
806605C0: 64646464.64646464 "dddddddd"

If the ASCII character that corresponds to the value contained in a byte is not printable, SDA displays a period ( . ). If the specified location does not exist in memory, SDA displays this message:

%SDA-E-NOTINPHYS, address : virtual data not in physical memory

To examine a range of locations, you can designate starting and ending locations separated by a colon. For example:

SDA> EXAMINE G40:G200

Alternatively, you can specify a location and a length, in bytes, separated by a semicolon. For example:

SDA> EXAMINE G400;16

When used to display the contents of a range of locations, the EXAMINE command displays six or ten columns of information. Ten columns are used if the terminal width is 132 or greater, or if a SET OUTPUT has been entered; six columns are used otherwise. An explanation of the columns is as follows:

• Each of the first four or eight columns represents a longword of memory, the contents of which are displayed in hexadecimal format.

• The fifth or ninth column lists the ASCII value of each byte in each longword displayed in the previous four or eight columns.

• The sixth or tenth column contains the address of the first, or rightmost, longword in each line. This address is also the address of the first, or leftmost, character in the ASCII representation of the longwords. Thus, you read the hexadecimal dump display from right to left, and the ASCII display from left to right.

If a series of virtual addresses does not exist in physical memory, SDA displays a message specifying the range of addresses that were not translated.

If a range of virtual locations contains only zeros, SDA displays this message:

Zeros suppressed from 'loc1' to 'loc2'

Decoding Locations
You can translate the contents of memory locations into instruction format by using the /INSTRUCTION qualifier. This qualifier causes SDA to display the location in symbolic notation (if possible) and its contents in instruction format. The operands of decoded instructions are also displayed in symbolic notation. The location must be longword aligned.
Examining Memory Regions
You can display an entire region of virtual memory by using one or more of the qualifiers /ALL, /SYSTEM, /P0, and /P1 with the EXAMINE command.

Other Uses
Other uses of the EXAMINE command appear in the following examples.

Examples

1. SDA> EXAMINE/PS 7FF95E78
   MBZ SPAL MBZ IPL VMM MBZ CURMOD INT PRVMOD
   0 00 0000000000 08 0 0 KERN 0 EXEC
   This example shows the display produced by the EXAMINE/PS command.

2. SDA> EXAMINE/PTE @QMMG$GQ_L1_BASE
   \n   3 3 2 2 2
   1 0 9 7 7 6 0
   0 8 6 5 0 7 6 0
   0 1 0 0 0 0 0 1
   Valid PTE: Read Prot = K---, Write Prot = K---
   Owner = K, Fault on = -E--, ASM = 00, Granularity Hint = 00
   CPY = 00 PFN = 00000C37
   The EXAMINE/PTE command displays and formats the level 1 page table entry at FFFFFFFF.FF7FC000.
EXIT

Exits from an SDA display or exits from the SDA utility.

Format

EXIT

Parameters

None.

Qualifiers

None.

Description

If SDA is displaying information on a video display terminal—and if that information extends beyond one screen—SDA displays a screen overflow prompt at the bottom of the screen:

Press RETURN for more.
SDA>

If you want to discontinue the current display at this point, enter the EXIT command. If you want SDA to execute another command, enter that command. SDA discontinues the display as if you entered EXIT, and then executes the command you entered.

When the SDA> prompt is not immediately preceded by the screen overflow prompt, entering EXIT causes your process to cease executing the SDA utility. When issued within a command procedure (either the SDA initialization file or a command procedure invoked with the execute command (@)), EXIT causes SDA to terminate execution of the procedure and return to the SDA prompt.
FORMAT

Displays a formatted list of the contents of a block of memory.

Format

FORMAT [/TYPE=block-type] location [/PHYSICAL]

Parameter

location
Location of the beginning of the data block. The location can be given as any valid SDA expression.

Qualifiers

/TYPE=block-type
Forces SDA to characterize and format a data block at location as the specified type of data structure. The /TYPE qualifier thus overrides the default behavior of the FORMAT command in determining the type and/or subtype of a data block, as described in the Description section. The block-type can be the symbolic prefix of any data structure defined by the operating system.

/PHYSICAL
 Specifies that the location given is a physical address.

Description

The FORMAT command performs the following actions:
• Characterizes a range of locations as a system data block
• Assigns, if possible, a symbol to each item of data within the block
• Displays all the data within the block

Most OpenVMS Alpha control blocks include two bytes that indicate the block type and/or subtype at offsets 0A16 and 0B16, respectively. The type and/or subtype associate the block with a set of symbols that have a common prefix. Each symbol's name describes a field within the block, and the value of the symbol represents the offset of the field within the block.

If the type and/or subtype bytes contain a valid block type/subtype combination, SDA retrieves the symbols associated with that type of block (see $DYNDEF) and uses their values to format the block.

For a given block type, all associated symbols have the form

<block_type>$<field>_<name>

where field is one of the following:
FORMAT

B  Byte
W  Word
L  Longword
Q  Quadword
O  Octaword
A  Address
C  Constant
G  Global Longword
P  Pointer
R  Structure (variable size)
T  Counted ASCII string (up to 31 characters)

If SDA cannot find the symbols associated with the block type specified in the block-type byte or by the /TYPE qualifier, it issues this message:

%SDA-E-NOSYMBOLS, no <block type> symbols found to format this block

If you receive this message, you may want to read additional symbols into the SDA symbol table and retry the FORMAT command. Many symbols that define OpenVMS Alpha data structures are contained within SDA$READ_DIR:SYSDEF.STB. Thus, you would issue the following command:

SDA> READ SDA$READ_DIR:SYSDEF.STB

If SDA issues the same message again, try reading additional symbols. Table 2–4 lists additional modules provided by the OpenVMS operating system. Alternatively, you can create your own object modules with the MACRO-32 Compiler for OpenVMS Alpha. See the READ command description for instructions on creating such an object module.

Certain OpenVMS Alpha data structures do not contain a block type and/or subtype. If bytes contain information other than a block type/subtype—or do not contain a valid block type/subtype—SDA either formats the block in a totally inappropriate way, based on the contents of offsets 0A_{16} and 0B_{16}, or displays this message:

%SDA-E-INVBLKTYP, invalid block type in specified block

To format such a block, you must reissue the FORMAT command, using the /TYPE qualifier to designate a block-type.

The FORMAT command produces a 3-column display:

- The first column shows the virtual address of each item within the block.
- The second column lists each symbolic name associated with a location within the block.
- The third column shows the contents of each item in hexadecimal format, including symbolization if a suitable symbol exists.
Example

SDA> READ SDA$READ_DIR:SYSDEF.STB
%SDA-I-READSYM, 913 symbols read from SYS$COMMON:[SYS$LDR]SYSDEF.STB
SDA> FORMAT G41F818

```
FFFFFFF.8041F818  UCB$L_FQFL 8041F818  UCB
   UCB$L_MB_MSGQFL
   UCB$L_RQFL
   UCB$W_MB_SEED
   UCB$W_UNIT_SEED

FFFFFFF.8041F81C  UCB$L_FQBL 8041F818  UCB
   UCB$L_MB_MSGQBL
   UCB$L_RQBL

FFFFFFF.8041F820  UCB$W_SIZE 0110
FFFFFFF.8041F822  UCB$B_TYPE 10
FFFFFFF.8041F823  UCB$B_FLCK 2C
FFFFFFF.8041F824  UCB$L_ASTQFL 00000000
   UCB$L_FPC
   UCB$L_MB_W_AST
   UCB$T_PARTNER

. . .
```

The READ command loads the symbols from SDA$READ_DIR:SYSDEF.STB into SDA's symbol table. The FORMAT command displays the data structure that begins at G41F818, a unit control block (UCB). If a field has more than one symbolic name, all such names are displayed. Thus, the field that starts at 8041F824 has four designations: UCB$L_ASTQFL, UCB$L_FPC, UCB$L_MB_W_AST, and UCB$T_PARTNER.

The contents of each field appear to the right of the symbolic name of the field. Thus, the contents of UCB$L_FQBL are 8041F818.
HELP

Displays information about the SDA utility, its operation, and the format of its commands.

Format

HELP  [topic-name]

Parameter

topic-name
Topic for which you need information. A topic can be a command name or one of the following keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYZE_USAGE_</td>
<td>Describes the parameters and qualifiers for the ANALYZE/CRASH_DUMP and</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>ANALYZE/SYSTEM DCL commands</td>
</tr>
<tr>
<td>CPU_CONTEXT</td>
<td>Describes the concept of CPU context as it governs the behavior of SDA</td>
</tr>
<tr>
<td>EXECUTE_COMMAND</td>
<td>Describes the use of @file to execute SDA commands contained in a file</td>
</tr>
<tr>
<td>EXPRESSIONS</td>
<td>Prints a description of SDA expressions</td>
</tr>
<tr>
<td>INITIALIZATION</td>
<td>Describes the circumstances under which SDA executes an initialization</td>
</tr>
<tr>
<td></td>
<td>file when first invoked</td>
</tr>
<tr>
<td>OPERATION</td>
<td>Describes how to operate SDA at your terminal and by means of the</td>
</tr>
<tr>
<td></td>
<td>site-specific startup procedure</td>
</tr>
<tr>
<td>PROCESS_CONTEXT</td>
<td>Describes the concept of process context as it governs the behavior of</td>
</tr>
<tr>
<td></td>
<td>SDA</td>
</tr>
<tr>
<td>SDA_CLUE_EXTENSION_</td>
<td>Provides an overview of SDA CLUE (Crash Log Utility Extractor)</td>
</tr>
<tr>
<td>COMMANDS</td>
<td></td>
</tr>
<tr>
<td>SDA_EXTENSION_ROUTINES</td>
<td>Describes how to write, debug, and invoke an SDA extension and provides</td>
</tr>
<tr>
<td></td>
<td>details of all callable routines</td>
</tr>
<tr>
<td>SDA_SPINLOCK_TRACING_COMMANDS</td>
<td>Provides an overview of SDA SPL (Spinlock Tracing Utility)</td>
</tr>
<tr>
<td>SYMBOLS</td>
<td>Describes the symbols used by SDA</td>
</tr>
</tbody>
</table>

Qualifiers

None.

Description

The HELP command displays brief descriptions of SDA commands and concepts on the terminal screen (or sends these descriptions to the file designated in a SET OUTPUT command). You can request additional information by specifying the name of a topic in response to the Topic? prompt.
If you do not specify a parameter in the HELP command, it lists the features of SDA and those commands and topics for which you can request help, as follows:

Example

SDA> HELP
HELP

The System Dump Analyzer (SDA) allows you to inspect the contents of memory as saved in the dump taken at crash time or as exists in a running system. You can use SDA interactively or in batch mode. You can send the output from SDA to a listing file. You can use SDA to perform the following operations:

- Assign a value to a symbol
- Examine memory of any process
- Format instructions and blocks of data
- Display device data structures
- Display memory management data structures
- Display a summary of all processes on the system
- Display the SDA symbol table
- Copy the system dump file
- Send output to a file or device
- Read global symbols from any object module
- Send output to a file or device
- Read global symbols from any object module
- Search memory for a given value

For help on performing these functions, use the HELP command and specify a topic.

Format

HELP [topic-name]

Additional information available:

- ANALYZE_Usage_Summary
- ATTACH
- CLUE
- COPY
- CPU_Context
- DEFINE
- DUMP
- EVALUATE
- EXAMINE
- Execute_Command
- EXIT
- Expressions
- FORMAT
- HELP
- Initialization
- MAP
- MODIFY
- Operation
- Process_Context
- READ
- REPEAT
- SDA_CLUE_Extension_Commands
- SDA_Extension_Routines
- SDA_Spinlock_Tracing_Commands
- SEARCH
- SET
- SHOW
- SPAWN
- SPL
- Symbols
- UNDEFINE
- VALIDATE
MAP

Transforms an address into an offset in a particular image.

Format

MAP address

Parameter

address
Address to be identified.

Qualifiers

None.

Description

The MAP command identifies the image name and offset corresponding to an address. With this information, you can examine the image map to locate the source module and program section offset corresponding to an address. MAP searches for the specified address in executive images first. It then checks activated images in process space to include those images installed using the /RESIDENT qualifier of the Install utility. Finally, it checks all image-resident sections in system space.

If the address cannot be found, MAP displays the following message:

%SDA-E-NOTINIMAGE, Address not within a system/installed image

Examples

1. SDA> MAP G90308

<table>
<thead>
<tr>
<th>Image</th>
<th>Base</th>
<th>End</th>
<th>Image Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS$VM</td>
<td>80090000</td>
<td>800ABA00</td>
<td>00000308</td>
</tr>
</tbody>
</table>

Examining the image map identified by this MAP command (SYS$VM.MAP) shows that image offset 308 falls within psect EXEC$HI_USE_PAGEABLE_CODE because the psect goes from offset 0 to offset 45D3:

EXEC$HI_USE_PAGEABLE_CODE 00000000 000045D3 000045D4 (17876.) 2 ** 5 . . .
SYSCREDDEL 00000000 0000149B 0000149C (5276.) 2 ** 5
SYSCRMFSC 000014A0 000045D3 00003134 (12596.) 2 ** 5
EXEC$NONPAGED_CODE 000045E0 000188B3 000172D4 (94932.) 2 ** 5 . . .
EXECUTE_FAULT 000045E0 0000483B 0000025C (604.) 2 ** 5
IOLOCK 00004840 000052E7 00000AA8 (2728.) 2 ** 5
LOCK_SYSTEM_PAGES . . .

Specifically, image offset 308 is located within source module SYSCREDDEL. Therefore, to locate the corresponding code, you would look in SYSCREDDEL for offset 308 in psect EXEC$HI_USE_PAGEABLE_CODE.
2. SDA> MAP G550000
   Image Base End Image Offset
   SYSDKDRIVER 80548000 80558000 00008000

   In this example, the MAP command identifies the address as an offset into an executive image that is not sliced. The base and end addresses are the boundaries of the image.

3. SDA> MAP G550034
   Image Base End Image Offset
   SYSDUDRIVER 80550000 80551400 00008034

   In this example, the MAP command identifies the address as an offset into an executive image that is sliced. The base and end addresses are the boundaries of the image section that contains the address of interest.

4. SDA> MAP GF0040
   Image Resident Section Base End Image Offset
   MAILSHR 800F0000 80119000 00000040

   The MAP command identifies the address as an offset into an image-resident section residing in system space.

5. SDA> MAP 12000
   Activated Image Base End Image Offset
   MAIL 00010000 000809FF 00002000

   The MAP command identifies the address as an offset into an activated image residing in process-private space.

6. SDA> MAP B2340
   Compressed Data Section Base End Image Offset
   LIBRTL 000B2000 000B6400 00080340

   The MAP command identifies the address as being within a compressed data section. When an image is installed with the Install utility using the /RESIDENT qualifier, the code sections are mapped in system space. The data sections are compressed into process-private space to reduce null pages or holes in the address space left by the absence of the code section. The SHOW PROCESS/IMAGE=ALL display shows how the data has been compressed; the MAP command searches this information to map an address in a compressed data section to an offset in an image.

7. SDA> MAP 7FC06000
   Shareable Address Data Section Base End Image Offset
   LIBRTL 7FC06000 7FC16800 00090000

   The MAP command identifies the address as an offset into a shareable address data section residing in P1 space.

8. SDA> MAP 7FC26000
   Read-Write Data Section Base End Image Offset
   LIBRTL 7FC26000 7FC27000 00080000

   The MAP command identifies the address as an offset into a read-write data section residing in P1 space.
9. **SDA> MAP 7FC36000**
   
<table>
<thead>
<tr>
<th>Shareable Read-Only Data Section</th>
<th>Base</th>
<th>End</th>
<th>Image Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRTL</td>
<td>7FC36000</td>
<td>7FC3F600</td>
<td>000C0000</td>
</tr>
</tbody>
</table>
   
   The MAP command identifies the address as an offset into a shareable read-only data section residing in P1 space.

10. **SDA> MAP 7FC56000**
    
    | Demand Zero Data Section      | Base   | End   | Image Offset |
    |-------------------------------|--------|-------|--------------|
    | LIBRTL                        | 7FC56000 | 7FC57000 | 000E0000     |
    
    The MAP command identifies the address as an offset into a demand zero data section residing in P1 space.
MODIFY DUMP

Allows a given byte, word, longword, or quadword in the dump to be modified.

Format

MODIFY DUMP {/BLOCK=n/OFFSET=n | /NEXT} [/CONFIRM=n]
{/BYTE | /WORD | /LONGWORD (d) | /QUADWORD} value

Parameter

value
The new value deposited in the specified location in the dump file.

Qualifiers

/BLOCK=n
Indicates block number to be modified. Required unless the /NEXT qualifier is given.

/OFFSET=n
Indicates byte offset within block to be modified. Required unless the /NEXT qualifier is given.

/CONFIRM=n
Checks existing contents of location to be modified.

/NEXT
Indicates that the byte(s) immediately following the location altered by the previous MODIFY DUMP command is/are to be modified. Used instead of the /BLOCK=n and /OFFSET=n qualifiers.

/BYTE
Indicates that only a single byte is to be replaced.

/WORD
Indicates that a word is to be replaced.

/LONGWORD
Indicates that a longword is to be replaced. This is the default.

/QUADWORD
Indicates that a quadword is to be replaced.

Description

The MODIFY DUMP command is used on a dump file that cannot be analyzed without specifying the /OVERRIDE qualifier on the ANALYZE/CRASH_DUMP command. The MODIFY DUMP command can be used to correct the problem that prevents normal analysis of a dump file. The MODIFY DUMP command can only be used when SDA has been invoked with the ANALYZE/CRASH_DUMP/OVERRIDE command.
Important

This command is not intended for general use. It is provided for the benefit of Compaq support personnel when investigating crash dumps that cannot be analyzed in other ways.

If the block being modified is part of either the dump header, the error log buffers, or the compression map, the changes made are not seen when you issue the appropriate SHOW DUMP command, unless you first exit from SDA and then reissue the ANALYZE/CRASH_DUMP command.

The MODIFY DUMP command sets a bit in the dump header to indicate that the dump has been modified. Subsequent ANALYZE/CRASH_DUMP commands issued to that file produce the following warning message:

%SDA-W-DUMPMOD, dump has been modified

Example

SDA>> MODIFY DUMP/BLOCK=10/OFFSET=100/WORD FF
This example shows the dump file modified with the word at offset 100 in block 00000010 replaced by 00FF.

SDA>> MODIFY DUMP/BLOCK=10/OFFSET=100/WORD 0/CONFIRM=EE
This example shows that the actual word value of 00FF at offset 100 in block 00000010 does not match the given value of 00EE. The following message is displayed:

%SDA-E-NOMATCH, expected value does not match value in dump; dump not updated

SDA>> MODIFY DUMP/BLOCK=10/OFFSET=100/WORD 0/CONFIRM=FF
This example shows the dump file modified with a word value of 00FF at offset 100 in block 00000010 replaced by 0000.
READ

Loads the global symbols contained in the specified file into the SDA symbol table.

Format

```
READ [/NO]LOG | /RELOCATE =expression | /SYMVA = expression |
   /EXECUTIVE [directory spec] | /FORCE filespec |
   /IMAGE filespec | filespec
```

Parameters

directory-spec
The directory-spec is the name of the directory containing the loadable images of the executive. This parameter defaults to SDA$READ_DIR which is a search list of SYS$LOADABLE_IMAGES and SYS$LIBRARY.

filespec
Name of the device, directory, and file that contains the file from which you want to read global symbols. The filespec defaults to SYS$DISK:[default-dir]filename.type, where SYS$DISK and [default-dir] represent the disk and directory specified in your last DCL command SET DEFAULT. If no type has been given in filespec, SDA first tries .STB and then .EXE.

If no device or directory is given in the file specification, and the file specification is not found in SYS$DISK:[default_dir], then SDA attempts to open the file SDA$READ_DIR:filename.type. If no type has been given in filespec, SDA first tries .STB and then .EXE.

If the file name is the same as that of an execlet or image, but the symbols in the file are not those of the execlet or image, then you must use the /FORCE qualifier, and optionally /RELOCATE and /SYMVA qualifiers, to tell SDA how to interpret the symbols in the file.

Qualifiers

/EXECUTIVE directory-spec
Reads into the SDA symbol table all global symbols and global entry points defined within all loadable images that make up the executive. For all the execlets in the system, SDA reads the .STB or .EXE files in the requested directory.

/FORCE filespec
Forces SDA to read the symbols file, regardless of what other information or qualifiers are specified. If you do not specify the /FORCE qualifier, SDA may not read the symbols file if the specified filespec matches the image name in either the executive loaded images or the current processes activated image list, and one of the following conditions is true:

- The image has a symbols vector (is a shareable image), and a symbols vector was not specified with the /SYMVA or /IMAGE qualifier.
- The image is sliced, and slicing information was not provided with the /IMAGE qualifier.
SDA Commands

READ

- The shareable or executive image is not loaded at the same address it was
  linked at, and the relocation information was not provided with either the
  /IMAGE or /RELOCATE qualifier.

The use of /FORCE [/SYMVA=addr]/[RELOCATE=addr] file spec is a variant of
the /IMAGE qualifier and avoids fixing up the symbols to match an image of the
same name.

/IMAGE filespec
 Searches the executive loaded image list and the current process activated image
list for the image specified by filespec. If the image is found, the symbols are
read in using the image symbol vector (if there is one) and either slicing or
relocation information.

This is the preferred way to read in the .STB files produced by the linker. These
.STB files contain all universal symbols, unless SYMBOL_TABLE=GLOBAL is
in the linker options file, in which case the .STB file contains all universal and
global symbols.

/LOG
/NOLOG
 The /LOG qualifier causes SDA to output the %SDA-I-READSYM message for
each symbol table file it reads. This is the default. The /LOG qualifier can be
specified with any other combination of parameters and qualifiers.

The /NOLOG qualifier suppresses the output of the %SDA-I-READSYM messages.
The /NOLOG qualifier can be specified with any other combination of parameters
and qualifiers.

/RELOCATE=expression
 Changes the relative addresses of the symbols to absolute addresses by adding
the value of expression to the value of each symbol in the symbol-table file to be
read. This qualifier changes those addresses to absolute addresses in the address
space into which the dump is mapped.

The relocation only applies to symbols with the relocate flag set. All universal
symbols must be found in the symbol vector for the image. All constants are read
in without any relocation.

If the image is sliced (image sections are placed in memory at different relative
offsets than how the image is linked), then the /RELOCATE qualifier does not
work. SDA compares the file name used as a parameter to the READ command
against all the image names in the executive loaded image list and the current
processes activated image list. If a match is found, and that image contains a
symbol vector, an error results. At this point you can either use the /FORCE
qualifier or the /IMAGE qualifier to override the error.

/SYMVA=expression
 Informs SDA whether the absolute symbol vector address is for a shareable
image (SYS$PUBLIC_VECTORS.EXE) or base system image (SYS$BASE_
IMAGE.EXE). All symbols found in the file with the universal flag are found by
referencing the symbol vector (that is, the symbol value is a symbol vector offset).
Description

The READ command symbolically identifies locations in memory and the definitions used by SDA for which the default files (SDA$READ_DIR:SYS$BASE_IMAGE.EXE and SDA$READ_DIR:REQSYSDEF.STB) provide no definition. In other words, the required global symbols are located in modules and symbol tables that have been compiled and/or linked separately from the executive. SDA extracts no local symbols from the files.

The file specified in the READ command can be the output of a compiler or assembler (for example, an .OBJ file).

\[\text{Note}\]

READ can read both OpenVMS VAX and OpenVMS Alpha format files. READ should not be used to read OpenVMS VAX format files that contain VAX specific symbols, as this might change the behavior of other OpenVMS Alpha SDA commands.

Most often the file is provided in SYS$LOADABLE IMAGES. Many SDA applications, for instance, need to load the definitions of system data structures by issuing a READ command specifying SYSDEF.STB. Others require the definitions of specific global entry points within the executive image.

The files in SYS$LOADABLE IMAGES define global locations within executive images, including those listed in Table 4-1. The actual list of executive images used varies, depending on platform type, devices, and the settings of several system parameters.

Table 4-1 Modules Defining Global Locations Within Executive Image

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME.EXE</td>
<td>$ACM system service</td>
</tr>
<tr>
<td>CNX$DEBUG.EXE</td>
<td>Connection Manager trace routines</td>
</tr>
<tr>
<td>DDIF$RMS_EXTENSION.EXE</td>
<td>Support for Digital Document Interchange Format (DDIF) file operations</td>
</tr>
<tr>
<td>ERRORLOG.STB</td>
<td>Error-logging routines and system services</td>
</tr>
<tr>
<td>EXCEPTION.STB(^1)</td>
<td>Bugcheck and exception-handling routines and those system services that declare condition and exit handlers</td>
</tr>
<tr>
<td>EXEC_INIT.STB</td>
<td>Initialization code</td>
</tr>
<tr>
<td>F11BXQP.STB</td>
<td>File system support</td>
</tr>
<tr>
<td>FC$GLOGALS.STB</td>
<td>Fibrechannel symbols</td>
</tr>
</tbody>
</table>

\(^1\)Variations of these files also exist, for example where the file name ends ".MON." System parameters such as SYSTEM_CHECK determine which image is loaded.

(continued on next page)
Table 4–1 (Cont.) Modules Defining Global Locations Within Executive Image

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGE_MANAGEMENT.STB</td>
<td>Image activator and the related system services</td>
</tr>
<tr>
<td>IO_ROUTINES.STB¹</td>
<td>$QIO system service, related system services (for example, $CANCEL and $ASSIGN), and supporting routines</td>
</tr>
<tr>
<td>LAT$RATING.EXE</td>
<td>CPU load balancing routines for LAT</td>
</tr>
<tr>
<td>LCK$DEBUG.EXE</td>
<td>Lock manager trace routines</td>
</tr>
<tr>
<td>LMF$GROUP_TABLE.EXE</td>
<td>Data structures for licensed product groups</td>
</tr>
<tr>
<td>LOCKING.STB</td>
<td>Lock management routines and system services</td>
</tr>
<tr>
<td>LOGICAL_NAMES.STB</td>
<td>Logical name routines and system services</td>
</tr>
<tr>
<td>MESSAGE_ROUTINES.STB</td>
<td>System message routines and system services (including $SNDJ BC and $GETTIM)</td>
</tr>
<tr>
<td>MSCP.EXE</td>
<td>Disk MSCP server</td>
</tr>
<tr>
<td>MULTIPATH.STB¹</td>
<td>Fibrechannel multipath support routines</td>
</tr>
<tr>
<td>NET$CSMACD.EXE</td>
<td>CSMA/CD LAN management module</td>
</tr>
<tr>
<td>NET$FDDI.EXE</td>
<td>FDDI LAN management module</td>
</tr>
<tr>
<td>NT_EXTENSION.EXE</td>
<td>NT extensions for persona system services</td>
</tr>
<tr>
<td>PROCESS_MANAGEMENT.STB¹</td>
<td>Scheduler, report system event, and supporting routines and system services</td>
</tr>
<tr>
<td>QSRV$GLOBALS.STB</td>
<td>$QIOserver symbols</td>
</tr>
<tr>
<td>RECOVERY_UNIT_SERVICES.STB</td>
<td>Recovery unit system services</td>
</tr>
<tr>
<td>RMS.EXE</td>
<td>Global symbols and entry points for RMS</td>
</tr>
<tr>
<td>SECURITY.STB¹</td>
<td>Security management routines and system services</td>
</tr>
<tr>
<td>SHELLxxK.STB</td>
<td>Process shell</td>
</tr>
<tr>
<td>SPL$DEBUG.EXE</td>
<td>Spinlock trace routines</td>
</tr>
<tr>
<td>SSPI.EXE</td>
<td>Security Support Provider Interface</td>
</tr>
<tr>
<td>SYS$xxDRIVER.EXE</td>
<td>Run-time device drivers</td>
</tr>
</tbody>
</table>

¹Variations of these files also exist, for example where the file name ends "_MON." System parameters such as SYSTEM_CHECK determine which image is loaded.

(continued on next page)
Table 4–1 (Cont.)  Modules Defining Global Locations Within Executive Image

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS$ATMWORKS351.EXE</td>
<td>PCI-ATM driver</td>
</tr>
<tr>
<td>SYS$CLUSTER.EXE</td>
<td>OpenVMS Cluster support routines</td>
</tr>
<tr>
<td>SYS$CPU_ROUTINES_xxxx.EXE</td>
<td>Processor-specific data and initialization routines</td>
</tr>
<tr>
<td>SYS$EW1000A.EXE</td>
<td>Gigabit Ethernet driver</td>
</tr>
<tr>
<td>SYS$GALAXY.STB</td>
<td>OpenVMS Galaxy support routines</td>
</tr>
<tr>
<td>SYS$IPC_SERVICES.EXE</td>
<td>Interprocess communication for DECdtm and Batch/Print</td>
</tr>
<tr>
<td>SYS$LAN.EXE</td>
<td>Common LAN routines</td>
</tr>
<tr>
<td>SYS$LAN_ATM.EXE</td>
<td>LAN routines for ATM</td>
</tr>
<tr>
<td>SYS$LAN_ATM4.EXE</td>
<td>LAN routines for ATM (ForeThought)</td>
</tr>
<tr>
<td>SYS$LAN_CSMACD.EXE</td>
<td>LAN routines for CSMA/CD</td>
</tr>
<tr>
<td>SYS$LAN_FDDI.EXE</td>
<td>LAN routines for FDDI</td>
</tr>
<tr>
<td>SYS$LAN_TR.EXE</td>
<td>LAN routines for Token Ring</td>
</tr>
<tr>
<td>SYS$MME_SERVICES.STB</td>
<td>Media Management Extensions</td>
</tr>
<tr>
<td>SYS$NETWORK_SERVICES.EXE</td>
<td>DECnet support</td>
</tr>
<tr>
<td>SYS$NTA.STB</td>
<td>NT affinity routines and services</td>
</tr>
<tr>
<td>SYS$PUBLIC_VECTORS.EXE</td>
<td>System service vector base image</td>
</tr>
<tr>
<td>SYS$QIOSERVER_KCLIENT.EXE</td>
<td>QIOserver client</td>
</tr>
<tr>
<td>SYS$QIOSERVER_KSERVER.EXE</td>
<td>QIOserver server</td>
</tr>
<tr>
<td>SYS$SCS.EXE</td>
<td>System Communication Services</td>
</tr>
<tr>
<td>SYS$TRANSACTION_SERVICES.EXE</td>
<td>DECdtm services</td>
</tr>
<tr>
<td>SYS$UTC_SERVICES.EXE</td>
<td>Universal Coordinated Time services</td>
</tr>
<tr>
<td>SYS$VCC.STB</td>
<td>Virtual I/O cache</td>
</tr>
<tr>
<td>SYS$VM.STB</td>
<td>System pager and swapper, along with their supporting routines, and management system services</td>
</tr>
<tr>
<td>SYS$XFCACHE.STB</td>
<td>Extented File Cache</td>
</tr>
<tr>
<td>SYS$DEVICE.STB</td>
<td>Mailbox driver and null driver</td>
</tr>
<tr>
<td>SYS$GETSYI.STB</td>
<td>Get System Information system service ($GETSYI)</td>
</tr>
<tr>
<td>SYS$SLDR_DYN.STB</td>
<td>Dynamic executive image loader</td>
</tr>
<tr>
<td>SYS/LICENSE.STB</td>
<td>Licensing system service ($LICENSE)</td>
</tr>
</tbody>
</table>

1Variations of these files also exist, for example where the file name ends "_MON." System parameters such as SYSTEM_CHECK determine which image is loaded.
2This file is located in SYS$LIBRARY.
Table 4–1 (Cont.) Modules Defining Global Locations Within Executive Image

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_DEBUG.EXE</td>
<td>XDelta and SCD routines</td>
</tr>
<tr>
<td>SYSTEM_PRIMITIVES.STB(^1)</td>
<td>Miscellaneous basic system routines, including those that allocate system memory, maintain system time, create fork processes, and control mutex acquisition</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION.STB(^1)</td>
<td>Routines that enforce synchronization</td>
</tr>
<tr>
<td>TCPPIP$BGDRIVER.STB(^3)</td>
<td>TCP/IP internet driver</td>
</tr>
<tr>
<td>TCPPIP$INETACP.STB(^3)</td>
<td>TCP/IP internet ACP</td>
</tr>
<tr>
<td>TCPPIP$INETDRIVER.STB(^3)</td>
<td>TCP/IP internet driver</td>
</tr>
<tr>
<td>TCPPIP$INTERNET_SERVICES.STB(^3)</td>
<td>TCP/IP internet executet</td>
</tr>
<tr>
<td>TCPPIP$NFS_SERVICES.STB(^3)</td>
<td>Symbols for the TCP/IP NFS server</td>
</tr>
<tr>
<td>TCPPIP$PROXY_SERVICES.STB(^3)</td>
<td>Symbols for the TCP/IP proxy executet</td>
</tr>
<tr>
<td>TCPPIP$PWIPACP.STB(^3)</td>
<td>TCP/IP PWIP ACP</td>
</tr>
<tr>
<td>TCPPIP$PWIPDRIVER.STB(^3)</td>
<td>TCP/IP PWIP driver</td>
</tr>
<tr>
<td>TCPPIP$TNDRIVER.STB(^3)</td>
<td>TCP/IP TELNET/RLOGIN server driver</td>
</tr>
<tr>
<td>TMSCP.EXE</td>
<td>Tape MSCP server</td>
</tr>
<tr>
<td>VMS_EXTENSION.EXE</td>
<td>VMS extensions for persona system services</td>
</tr>
</tbody>
</table>

\(^1\)Variations of these files also exist, for example where the file name ends ".MON." System parameters such as SYSTEM_CHECK determine which image is loaded.

\(^3\)Only available if TCP/IP has been installed. These are found in SYS$SYSTEM, and are not automatically read in when you issue a READ/EXEC command.

SDA can also read symbols from an image .EXE or .STB produced by the linker. The STB and EXE files only contain universal symbols. The STB file, however, can be forced to have global symbols for the image if you use the SYMBOL_TABLE=GLOBAL option in the linker options file.

A number of ready-built symbol table files ship with OpenVMS Alpha. They can be found in the directory SYS$LOADABLE_IMAGES, and all have names of the form xyzDEF.STB. Of these files, SDA automatically reads REQSYSDEF.STB on activation. You can add the symbols in the other files to SDA's symbol table using the READ command. Table 2–4 lists the files that OpenVMS Alpha provides in SYS$LOADABLE_IMAGES that define data structure offsets.

The following MACRO program, GLOBALS.MAR, shows how to obtain symbols in addition to those in SYS$BASE_IMAGE.EXE, other executive images listed in Table 4–1, and the symbol table files that are listed in Table 2–4.
.TITLE GLOBALS

$PHDDEF GLOBAL ; Process header definitions
$DDBDEF GLOBAL ; Device data block
$UCBDEF GLOBAL ; Unit control block
$VCBDEF GLOBAL ; Volume control block
$ACBDEF GLOBAL ; AST control block
$IRPDEF GLOBAL ; I/O request packet
; more can be inserted here
.END

Use the command below to generate an object module file containing the globals defined in the program.

$MACRO GLOBALS+SYS$LIBRARY:LIB/LIBRARY /OBJECT=GLOBALS.STB

Examples

1. SDA> READ SDA$READ_DIR:SYSDEF.STB
   %SDA-I-READSYM, 10010 symbols read from SYS$COMMON:[SYSEXE]SYSDEF.STB;1

   The READ command causes SDA to add all the global symbols in SDA$READ_DIR:SYSDEF.STB to the SDA symbol table. Such symbols are useful when you are formatting an I/O data structure, such as a unit control block or an I/O request packet.

2. SDA> SHOW STACK
   Process stacks (on CPU 00)
   -------------------------
   Current operating stack (KERNEL):
   00000000.7FF95CD0 FFFFFFFF.80430CE0 SCH$STATE_TO_COM+00040
   00000000.7FF95CD8 00000000.00000000
   00000000.7FF95CE0 FFFFFFFF.81E9CB04 LNM$SEARCH_ONE_C+000E4
   00000000.7FF95CE8 FFFFFFFF.8007A988 PROCESS_MANAGEMENT_NPRO+0E988
   SP =>00000000.7FF95CF0 00000000.00000000
   00000000.7FF95CF8 00000000.006080C1
   00000000.7FF95D00 FFFFFFFF.80501FDC
   00000000.7FF95D08 FFFFFFFF.81A5B720

   SDA> READ/IMAGE SYS$LOADABLE_IMAGES:PROCESS_MANAGEMENT
   %SDA-I-READSYM, 767 symbols read from SYS$COMMON:[SYS$LDR]PROCESS_MANAGEMENT.STB;1
   SDA> SHOW STACK
   Process stacks (on CPU 00)
   -------------------------
   Current operating stack (KERNEL):
   00000000.7FF95CD0 FFFFFFFF.80430CE0 SCH$FIND_NEXT_PROC
   00000000.7FF95CD8 00000000.00000000
   00000000.7FF95CE0 FFFFFFFF.81E9CB04 LNM$SEARCH_ONE_C+000E4
   00000000.7FF95CE8 FFFFFFFF.8007A988 SCH$INTERRUPT+00068
   SP =>00000000.7FF95CF0 00000000.00000000
   00000000.7FF95CF8 00000000.006080C1
   00000000.7FF95D00 FFFFFFFF.80501FDC
   00000000.7FF95D08 FFFFFFFF.81A5B720

   The initial SHOW STACK command contains an address that SDA resolves into an offset from the PROCESS_MANAGEMENT executive image. The READ command loads the corresponding symbols into the SDA symbol table such that

SDA Commands 4–39
the reissue of the SHOW STACK command subsequently identifies the same location as an offset within a specific process management routine.
REPEAT

Repeats execution of the last command issued. On terminal devices, the KP0 key performs the same function as the REPEAT command with no parameter or qualifier.

Format

REPEAT [count | /UNTIL=condition]

Parameter

count
The number of times the previous command is to be repeated. The default is a single repeat.

Qualifier

/UNTIL=condition
Defines a condition that terminates the REPEAT command. By default, there is no terminating condition.

Description

The REPEAT command is useful for stepping through a linked list of data structures, or for examining a sequence of memory locations. When used with ANALYZE/SYSTEM, it allows the changing state of a system location or data structure to be monitored.

Examples

1. SDA> SPAWN CREATE SDATEMP.COM
   SEARCH 0:FFFFFFF 12345678
   SET PROCESS/NEXT
   ^Z
   SDA> SET PROCESS NULL
   SDA> &SDATEMP
   SDA> REPEAT/UNTIL = BADROC

   This example demonstrates how the address space of each process in a system or dump can be searched for a given pattern.

2. SDA> SHOW CALL_FRAME
   Call Frame Information
   --------------------------
   Stack Frame Procedure Descriptor
   Flags: Base Register = FP, Jacket, Native
   Procedure Entry: FFFFF0.80080CE0
   Return address on stack = FFFFFFF.8004CF30
The first SHOW CALL_FRAME displays the call frame indicated by the current FP value. Because the /NEXT_FP qualifier to the instruction displays the call frame indicated by the saved FP in the current call frame, you can use the REPEAT command to repeat the SHOW CALL_FRAME/NEXT_FP command and follow a chain of call frames.
SEARCH

Scans a range of memory locations for all occurrences of a specified value.

Format

SEARCH [/qualifier] range [=] expression

Parameters

range
Location in memory to be searched. A location can be represented by any valid SDA expression. To search a range of locations, use the following syntax:

\[ m:n \]

Range of locations to be searched, from \( m \) to \( n \)

\[ m;n \]

Range of locations to be searched, starting at \( m \) and continuing for \( n \) bytes

expression
The value for which SDA is to search. SDA evaluates the expression and searches the specified range of memory for the resulting value. For a description of SDA expressions, see Section 2.6.2.

If you do not use an equals sign to separate range and expression, then you must insert a space between them.

Qualifiers

/LENGTH={QUADWORD|LONGWORD|WORD|BYTE}
Specifies the size of the expression value that the SEARCH command uses for matching. If you do not specify the /LENGTH qualifier, the SEARCH command uses a longword length by default.

/MASK=n
Allows the SEARCH command finer granularity in its matches. It compares only the given bits of a byte, word, longword, or quadword. To compare bits when matching, you set the bits in the mask; to ignore bits when matching, you clear the bits in the mask.

/STEPS={QUADWORD|LONGWORD|WORD|BYTE|value}
Specifies the step factor of the search through the specified memory range. After the SEARCH command has performed the comparison between the value of expression and memory location, it adds the specified step factor to the address of the memory location. The resulting location is the next location to undergo the comparison. If you do not specify the /STEPS qualifier, the SEARCH command uses a step factor of a longword.

/PHYSICAL
Specifies that the addresses used to define the range of locations to be searched are physical addresses.

Description

SEARCH displays each location as each value is found. If you press Ctrl/T while using the SEARCH command, the system displays how far the search has progressed. The progress display is always output to the terminal even if a SET OUTPUT <file> command has previously been entered.
**SDA Commands**

**SEARCH**

**Examples**

1. **SDA> SEARCH GB81F0;500 B41B0000**
   Searching from FFFFFFFF.800B81F0 to FFFFFFFF.800B86EF in LONGWORD steps for B41B0000...
   Match at FFFFFFFF.800B86E4 B41B0000
   
   This SEARCH command finds the value B41B0000 in the longword at FFFFFFFF.800B86E4.

2. **SDA> SEARCH 80000000;200/STEPS=BYTE 82**
   Searching from FFFFFFFF.80000000 to FFFFFFFF.800001FF in BYTE steps for 00000082...
   Match at FFFFFFFF.8000012C 00000082
   
   This SEARCH command finds the value 00000082 in the longword at FFFFFFFF.8000012C.

3. **SDA> SEARCH/LENGTH=WORD 80000000;100 10**
   Match at FFFFFFFF.80000030 0010
   Match at FFFFFFFF.80000040 0010
   Match at FFFFFFFF.80000090 0010
   Match at FFFFFFFF.800000A0 0010
   Match at FFFFFFFF.800000C0 0010
   5 matches found
   
   This SEARCH command finds the value 0010 in the words at FFFFFFFF.80000030, FFFFFFFF.80000040, FFFFFFFF.80000090, FFFFFFFF.800000A0, FFFFFFFF.800000C0.

4. **SDA> SEARCH/MASK=FF000000 80000000;40 20000000**
   Searching from FFFFFFFF.80000000 to FFFFFFFF.8000003F in LONGWORD steps for 20000000...
   (Using search mask of FF000000)
   Match at FFFFFFFF.80000000 201F0104
   Match at FFFFFFFF.80000010 201F0001
   2 matches found
   
   This SEARCH command finds the value 20 in the upper byte of the longwords at FFFFFFFF.80000000 and FFFFFFFF.80000010, regardless of the contents of the lower three bytes.
SET CPU

When analyzing a system dump, selects a processor to become the current CPU for SDA. (This command cannot be used when analyzing the running system.)

Format

SET CPU cpu-id

Parameter

cpu-id

Numeric value from 00_{16} to 1F_{16} indicating the identity of the processor to be made the current CPU. If you specify a value outside this range or a cpu-id of a processor that was not active at the time of the system failure, SDA displays the following message:

%SDA-E-CPUNOTVLD, CPU not booted or CPU number out of range

Qualifiers

None.

Description

When you invoke SDA to examine a system dump, the current CPU context for SDA defaults to that of the processor that caused the system to fail. When analyzing a system failure from a multiprocessing system, you may find it useful to examine the context of another processor in the configuration.

The SET CPU command changes the current CPU context for SDA to that of the processor indicated by cpu-id. The CPU specified by this command becomes the current CPU for SDA until you either exit from SDA or change the CPU context for SDA by issuing one of the following commands:

- SET CPU cpu-id
- SHOW CPU cpu-id
- SHOW CRASH
- SHOW MACHINE_CHECK cpu-id

The following commands also change the CPU context for SDA if the process-name, pcb-address, or index number (nn) refers to a current process:

- SET PROCESS process-name
- SET PROCESS/ADDRESS=pcb-address
- SET PROCESS/INDEX=nn
- SHOW PROCESS process-name
- SHOW PROCESS/ADDRESS=pcb-address
- SHOW PROCESS/INDEX=nn
Changing CPU context can cause an implicit change in process context under the following circumstances:

- If there is a current process on the CPU made current, SDA changes its process context to that of that CPU’s current process.
- If there is no current process on the CPU made current, the SDA process context is undefined and no process-specific information is available until you set the SDA process context to that of a specific process.

See Chapter 2, Section 2.5 for further discussion of the way in which SDA maintains its context information.
SET ERASE_SCREEN

Enables or disables the automatic clearing of the screen before each new page of SDA output.

Format

SET ERASE_SCREEN {ON | OFF}

Parameters

ON
Enables the screen to be erased before SDA outputs a new heading. This setting is the default.

OFF
Disables the erasing of the screen.

Qualifiers

None.

Description

SDA’s usual behavior is to erase the screen and then show the data. By setting the OFF parameter, the clear screen action is replaced by a blank line. This action does not affect what is written to a file when the SET LOG or SET OUTPUT commands are used.

Examples

1. SDA> SET ERASE_SCREEN ON
   The clear screen action is now enabled.

2. SDA> SET ERASE_SCREEN OFF
   The clear screen action is disabled.
SET FETCH

Sets the default size and access method of address data used when SDA evaluates an expression that includes the @ unary operator.

Format

SET FETCH  [{QUADWORD | LONGWORD | WORD | BYTE}][,][{PHYSICAL | VIRTUAL}]

Parameters

QUADWORD
Sets the default size to 8 bytes.

LONGWORD
Sets the default size to 4 bytes.

WORD
Sets the default size to 2 bytes.

BYTE
Sets the default size to 1 byte.

PHYSICAL
Sets the default access method to physical addresses.

VIRTUAL
Sets the default access method to virtual addresses.

Note that you can specify one and only one parameter out of each group. If you are changing both size and access method, separate the two parameters by spaces and/or a comma. Include a comma only if you are specifying a parameter from both groups. See examples 5 and 6.

Qualifiers

None.

Description

Sets the default size and/or default access method of address data used by the @ unary operator in commands such as EXAMINE and EVALUATE. SDA uses the current default size unless it is overridden by the ^Q, ^L, ^W, or ^B qualifier on the @ unary operator in an expression. SDA uses the current default access method unless it is overridden by the ^P or ^V qualifier on the @ unary operator in an expression.

Examples

1. SDA> EXAMINE MMG$GQ_SHARED_VA_PTES
   MMG$GQ_SHARED_VA_PTES: FFFFFFFD.FF7FE000 "a......"

   This shows the location’s contents of a 64-bit virtual address.
2. **SDA** > **SET FETCH LONG**  
**SDA** > **EXAMINE @MMGSQ$Q_SHARED_VA_PTES**  
%SDA-E-NOTINPHYS, FFFFFF.FF7FE000 : virtual data not in physical memory  
This shows a failure because the **SET FETCH LONG** causes **SDA** to assume that it should take the lower 32 bits of the location’s contents as a longword value, sign-extend them, and use that value as an address.

3. **SDA** > **EXAMINE @^QMMGSQ$Q_SHARED_VA_PTES**  
FFFFFD.FF7FE000: 000001D0.40001119 "@..."  
This shows the correct results by overriding the **SET FETCH LONG** with the ^Q qualifier on the @ operator. **SDA** takes the full 64-bits of the location’s contents and uses that value as an address.

4. **SDA** > **SET FETCH QUAD**  
**SDA** > **EXAMINE @MMGSQ$Q_SHARED_VA_PTES**  
FFFFFD.FF7FE000: 000001D0.40001119 "@..."  
This shows the correct results by changing the default fetch size to a quadword.

5. **SDA** > **SET FETCH PHYSICAL**  
**SDA** > **EXAMINE /PHYSICAL @0**  
This command uses the contents of the physical location 0 as the physical address of the location to be examined.

6. **SDA** > **SET FETCH QUADWORD, PHYSICAL**  
This command sets the default fetch size and default access method at the same time.
SET LOG

Initiates or discontinues the recording of an SDA session in a text file.

Format

SET [NO]LOG  filespec

Parameter

filespec
Name of the file in which you want SDA to log your commands and their output. The default filespec is SYS$DISK:[default_dir]filename.LOG, where SYS$DISK and [default-dir] represent the disk and directory specified in your last DCL command SET DEFAULT. You must specify a file name.

Qualifiers

None.

Description

The SET LOG command echoes the commands and output of an SDA session to a log file. The SET NOLOG command terminates this behavior.

The following differences exist between the SET LOG command and the SET OUTPUT command:

• When logging is in effect, your commands and their results are still displayed on your terminal. The SET OUTPUT command causes the displays to be redirected to the output file and they no longer appear on the screen.

• If an SDA command requires that you press Return to produce successive screens of display, the log file produced by SET LOG will record only those screens that are actually displayed. SET OUTPUT, however, sends the entire output of any SDA commands to its listing file.

• The SET LOG command produces a log file with a default file type of .LOG; the SET OUTPUT command produces a listing file whose default file type is .LIS.

• The SET LOG command does not record output from the HELP command in its log file. The SET OUTPUT command can record HELP output in its listing file.

• The SET OUTPUT command can generate a table of contents, each item of which refers to a display written to its listing file. SET OUTPUT also produces running heads for each page of output. The SET LOG command does not produce these items in its log file.

Note that, if you use the SET OUTPUT command to redirect output to a listing file, a SET LOG command to direct the same output to a log file is ineffective until output is restored to the terminal.
SET OUTPUT

Redirects output from SDA to the specified file or device.

Format

SET OUTPUT [/INDEX | /NOINDEX | /HEADER | /NOHEADER | /SINGLE_COMMAND] filespec

Parameter

filespec
Name of the file to which SDA is to send the output generated by its commands. The default filespec is SYS$DISK:[default_dir]filename.LIS, where SYS$DISK and [default-dir] represent the disk and directory specified in your last DCL command SET DEFAULT. You must specify a file name.

Qualifiers

/INDEX
/NOINDEX
The /INDEX qualifier causes SDA to include an index page at the beginning of the output file. This is the default, unless /NOHEADER is specified; see the /NOHEADER description. The /NOINDEX qualifier causes SDA to omit the index page from the output file.

/HEADER
/NOHEADER
The /HEADER qualifier causes SDA to include a heading at the top of each page of the output file. This is the default. The /NOHEADER qualifier causes SDA to omit the page headings. Use of /NOHEADER implies /NOINDEX.

/SINGLE_COMMAND
Indicates to SDA that the output for a single command is to be written to the specified file and that subsequent output should be written to the terminal.

Description

When you use the SET OUTPUT command to send the SDA output to a file or device, SDA continues displaying the SDA commands that you enter but sends the output generated by those commands to the file or device you specify. (See the description of the SET LOG command for a list of differences between the SET LOG and SET OUTPUT commands.)

When you finish directing SDA commands to an output file and want to return to interactive display, issue the following command:

SDA> SET OUTPUT SYS$OUTPUT

Note that this command is not needed when the /SINGLE_COMMAND qualifier was specified on the original SET OUTPUT command.

If you use the SET OUTPUT command to send the SDA output to a listing file and do not specify /NOINDEX or /NOHEADER, SDA builds a table of contents that identifies the displays you selected and places the table of contents at the beginning of the output file. The SET OUTPUT command formats the output into pages and produces a running head at the top of each page, unless you specify /NOHEADER.
Note

See the description of the DUMP command for use of SET OUTPUT/NOHEADER.
SET PROCESS

Selects a process to become the SDA current process.

Format

SET PROCESS {/ADDRESS=pcb-address | process-name | /ID=nn |
/NINDEX=nn | /NEXT | /SYSTEM}

Parameter

process-name
Name of the process to become the SDA current process. The process-name is a string containing up to 15 uppercase or lowercase characters; numerals, the dollar sign ($), and the underscore (_) can also be included in the string. If you include characters other than these, you must enclose the entire string in quotation marks (" ").

Qualifiers

/ADDRESS=pcb-address
Specifies the process control block (PCB) address of a process in order to display information about the process.

/ID=nn

/INFO=nn
Specifies the process for which information is to be displayed by its index into the system's list of software process control blocks (PCBs), or by its process identification. You can supply the following values for nn:

- The process index itself.
- The process identification (PID) or extended PID longword, from which SDA extracts the correct index. The PID or extended PID of any thread of a process with multiple kernel threads may be specified. Any thread-specific data displayed by further commands will be for the given thread.

To obtain these values for any given process, issue the SDA command SHOW SUMMARY/THREADS. The /ID=nn and /INDEX=nn qualifiers can be used interchangeably.

/NEXT
Causes SDA to locate the next valid process in the process list and select that process. If there are no further valid processes in the process list, SDA returns an error.

/SYSTEM
Specifies the new current process by the system process control block (PCB). The system PCB and process header (PHD) parallel the data structures that describe processes. They contain the system working set list, global section table, and other systemwide data.
SDA Commands
SET PROCESS

Description

When you issue an SDA command such as EXAMINE, SDA displays the contents of memory locations in its current process. To display any information about another process, you must change the current process with the SET PROCESS command.

When you invoke SDA to analyze a crash dump, the process context defaults to that of the process that was current at the time of the system failure. If the failure occurred on a multiprocessing system, SDA sets the CPU context to that of the processor that caused the system to fail. The process context is set to that of the process that was current on that processor.

When you invoke SDA to analyze a running system, its process context defaults to that of the current process, that is, the one executing SDA.

The SET PROCESS command changes the current SDA process context to that of the process indicated by process-name, pcb-address, or /INDEX=nn. The process specified by this command becomes the current process for SDA until you either exit from SDA or change SDA process context by issuing one of the following commands:

```
SET PROCESS process-name
SET PROCESS/ADDRESS=pcb-address
SET PROCESS/INDEX=nn
SET PROCESS/SYSTEM
SHOW PROCESS process-name
SHOW PROCESS/ADDRESS=pcb-address
SHOW PROCESS/INDEX=nn
SHOW PROCESS/SYSTEM
```

When you analyze a crash dump from a multiprocessing system, changing process context may require a switch of CPU context as well. For instance, if you issue a SET PROCESS command for a process that is current on another CPU, SDA automatically changes its CPU context to that of the CPU on which that process is current. The following commands can have this effect if process-name, pcb-address, or index number (nn) refers to a current process:

```
SET PROCESS process-name
SET PROCESS/ADDRESS=pcb-address
SET PROCESS/INDEX=nn
SHOW PROCESS process-name
SHOW PROCESS/ADDRESS=pcb-address
SHOW PROCESS/INDEX=nn
```

The following commands will also switch process context when analyzing a system dump, if there was a current process on the target CPU at the time of the crash:

```
SET CPU cpu-id
SHOW CPU cpu-id
SHOW CRASH
SHOW MACHINE_CHECK cpu-id
```

See Chapter 2, Section 2.5 for further discussion of the way in which SDA maintains its context information.
Example

SDA> SHOW PROCESS
Process index: 0012  Name: ERRFMT  Extended PID: 00000052
-----------------------------------------------------------
Process status:  02040001  RES,PHDRES,INTER
status2:  00000001  QUANTUM_RESCHED
PCB address  80D772C0  JIB address  80556600
PHD address  80477200  Swapfile disk address  01000F01
HTB vector address  80D775AC  HWPCB address  81260080
Callback vector address  00000000  Termination mailbox  0000
Master internal PID  00100004  Subprocess count  0
Creator extended PID  00000000  Creator internal PID  00000000
Previous CPU Id  00000000  Current CPU Id  00000000
Previous ASNSEQ  0000000000000001  Previous ASN  0000000000000002E
Initial process priority  4  Delete pending count  0
# open files allowed left  100  Direct I/O count/limit  150/150
UIC [00001,000004]  Buffered I/O count/limit  149/150
Abs time of last event  0069D34E  BUFIO byte count/limit  99424/99808
ASTs remaining  247  # of threads  1
Swapped copy of LEFC0  00000000  Timer entries allowed left  63
Swapped copy of LEFC1  00000000  Active page table count  4
Global cluster 2 pointer  00000000  Process WS page count  32
Global cluster 3 pointer  00000000  Global WS page count  31

This SHOW PROCESS command shows the current process to be ERRFMT, and displays information from its PCB and job information block (JIB).

See the description of the REPEAT command for an example of the use of the SET PROCESS/NEXT command.
SDA Commands
SET RMS

SET RMS

Changes the options shown by the SHOW PROCESS/RMS command.

Format

SET RMS = (option[,...])

Parameter

option
Data structure or other information to be displayed by the SHOW PROCESS/RMS command. Table 4–2 lists those keywords that may be used as options.

Table 4–2 SET RMS Command Keywords for Displaying Process RMS Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[NO]ALL[:ifi]¹</td>
<td>All control blocks (default)</td>
</tr>
<tr>
<td>[NO]ASB</td>
<td>Asynchronous save block</td>
</tr>
<tr>
<td>[NO]BDB</td>
<td>Buffer descriptor block</td>
</tr>
<tr>
<td>[NO]BDBSUM</td>
<td>BDB summary page</td>
</tr>
<tr>
<td>[NO]BLB</td>
<td>Buffer lock block</td>
</tr>
<tr>
<td>[NO]BLBSUM</td>
<td>Buffer lock summary page</td>
</tr>
<tr>
<td>[NO]CCB</td>
<td>Channel control block</td>
</tr>
<tr>
<td>[NO]DRC</td>
<td>Directory cache</td>
</tr>
<tr>
<td>[NO]FAB</td>
<td>File access block</td>
</tr>
<tr>
<td>[NO]FCB</td>
<td>File control block</td>
</tr>
<tr>
<td>[NO]FSB</td>
<td>File statistics block</td>
</tr>
<tr>
<td>[NO]FWA</td>
<td>File work area</td>
</tr>
<tr>
<td>[NO]GBD</td>
<td>Global buffer descriptor</td>
</tr>
<tr>
<td>[NO]GBDSUM</td>
<td>GBD summary page</td>
</tr>
<tr>
<td>[NO]GBH</td>
<td>Global buffer header</td>
</tr>
<tr>
<td>[NO]GBHSH</td>
<td>Global buffer hash table</td>
</tr>
<tr>
<td>[NO]GBSB</td>
<td>Global buffer synchronization block</td>
</tr>
<tr>
<td>[NO]IDX</td>
<td>Index descriptor</td>
</tr>
<tr>
<td>[NO]IFAB[:ifi]¹</td>
<td>Internal FAB</td>
</tr>
<tr>
<td>[NO]IFB[:ifi]¹</td>
<td>Internal FAB</td>
</tr>
<tr>
<td>[NO]IRAB</td>
<td>Internal RAB</td>
</tr>
<tr>
<td>[NO]IRB</td>
<td>Internal RAB</td>
</tr>
<tr>
<td>[NO]FB</td>
<td>Journaling file block</td>
</tr>
<tr>
<td>[NO]KLTB</td>
<td>Key-less-than block</td>
</tr>
</tbody>
</table>

¹The optional parameter ifi is an internal file identifier. The default ifi (ALL) is all the files the current process has opened.

(continued on next page)
Table 4–2 (Cont.)  SET RMS Command Keywords for Displaying Process RMS Information

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[NO]NAM</td>
<td>Name block</td>
</tr>
<tr>
<td>[NO]NWA</td>
<td>Network work area</td>
</tr>
<tr>
<td>[NO]PIO</td>
<td>Process-permanent I/O data structures used instead of process image data structures</td>
</tr>
<tr>
<td>[NO]RAB</td>
<td>Record access block</td>
</tr>
<tr>
<td>[NO]RLB</td>
<td>Record lock block</td>
</tr>
<tr>
<td>[NO]RU</td>
<td>Recovery unit structures, including the recovery unit block (RUB), recovery unit stream block (RUSB), and recovery unit file block (RUFB)</td>
</tr>
<tr>
<td>[NO]SFSB</td>
<td>Shared file synchronization block</td>
</tr>
<tr>
<td>[NO]WCB</td>
<td>Window control block</td>
</tr>
<tr>
<td>[NO]XAB</td>
<td>Extended attribute block</td>
</tr>
<tr>
<td>[NO]*</td>
<td>Current list of options displayed by the SHOW RMS command</td>
</tr>
</tbody>
</table>

The default **option** is **option=(ALL,NOPIO)**, designating for display by the SHOW PROCESS/RMS command all structures for all files related to the process image I/O.

To list more than one option, enclose the list in parentheses and separate options by commas. You can add a given data structure to those displayed by ensuring that the list of keywords begins with the asterisk (*) symbol. You can delete a given data structure from the current display by preceding its keyword with “NO.”

**Qualifiers**

None.

**Description**

The SET RMS command determines the data structures to be displayed by the SHOW PROCESS/RMS command. (See the examples included in the discussion of the SHOW PROCESS command for information provided by various displays.) You can examine the options that are currently selected by issuing a SHOW RMS command.
SDA Commands

SET RMS

Examples

1. SDA> SHOW RMS
   RMS Display Options: IFB, IRB, IDX, BDB, BDBSUM, ASB, CCB, WCB, FCB, FAB, RAB, NAM, XAB, RLB, BLB, BLBSUM, GBD, GBH, FWA, GBDSUM, JFB, NWA, RU, DRC, SFSB, GBSB

   Display RMS structures for all IFI values.
   SDA> SET RMS=IFB
   SDA> SHOW RMS

   RMS Display Options: IFB

   Display RMS structures for all IFI values.

   The first SHOW RMS command shows the default selection of data structures that are displayed in response to a SHOW PROCESS/RMS command. The SET RMS command selects only the IFB to be displayed by subsequent SET/PROCESS commands.

2. SDA> SET RMS=(*,BLB, BLBSUM, RLB)
   SDA> SHOW RMS

   RMS Display Options: IFB, RLB, BLB, BLBSUM

   Display RMS structures for all IFI values.

   The SET RMS command adds the BLB, BLBSUM, and RLB to the list of data structures currently displayed by the SHOW PROCESS/RMS command.

3. SDA> SET RMS=(*, NORLB, IFB:05)
   SDA> SHOW RMS

   RMS Display Options: IFB, BLB, BLBSUM

   Display RMS structures only for IFI=5.

   The SET RMS command removes the RLB from those data structures displayed by the SHOW PROCESS/RMS command and causes only information about the file with the \texttt{ifi} of 5 to be displayed.

4. SDA> SET RMS=(*, PIO)

   The SET RMS command indicates that the data structures designated for display by SHOW PROCESS/RMS be associated with process-permanent I/O instead of image I/O.
SET SIGN_EXTEND

Enables or disables the sign extension of 32-bit addresses.

Format

SET SIGN_EXTEND {ON | OFF}

Parameters

on
Enables automatic sign extension of 32-bit addresses with bit 31 set. This is the default.

off
Disables automatic sign extension of 32-bit addresses with bit 31 set.

Qualifiers

None.

Description

The 32-bit S0/S1 addresses need to be sign-extended to access 64-bit S0/S1 space. To do this, specify explicitly sign-extended addresses, or set the sign-extend command to on, which is the default.

However, to access addresses in P2 space, addresses must not be sign-extended. To do this, specify a zero in front of the address, or set the sign-extend command to off.

Examples

1. SDA> set sign_extend on
   SDA> examine 80400000
   FFFFFFFFF.80400000: 23DEFF90.4A607621
   
   This shows the SET SIGN_EXTEND command as ON.

2. SDA> set sign_extend off
   SDA> examine 80400000
   %SDA-E-NOTINPHYS, 00000000.80400000: virtual data not in physical memory
   
   This shows the SET SIGN_EXTEND command as OFF.
SET SYMBOLIZE

Enables or disables symbolization of addresses in the display from an EXAMINE command.

Format

SET SYMBOLIZE {ON | OFF}

Parameters

ON
Enables symbolization of addresses.

OFF
Disables symbolization of addresses.

Qualifiers

None.

Examples

1. SDA> SET SYMBOLIZE ON
SDA> examine g1234
SYS$PUBLIC_VECTORS+01234: 47DF041C "..AG"

2. SDA> SET SYMBOLIZE OFF
SDA> examine g1234
FFFFFFFF.80001234: 47DF041C "..AG"

This example shows the effect of enabling (default) or disabling symbolization of addresses.
SHOW ADDRESS

Displays the page table related information about a memory address.

Format

SHOW ADDRESS address [/PHYSICAL]

Parameter

address
Displays the requested address.

Qualifier

/PHYSICAL
Indicates that a physical address has been given. The SHOW ADDRESS command displays the virtual address that maps to the given physical address.

Description

The SHOW ADDRESS command displays the region of memory that contains the memory address. It also shows all the page table entries (PTEs) that map the page and can show the range of addresses mapped by the given address if it is the address of a PTE.

When the /PHYSICAL qualifier is given, the SHOW ADDRESS command displays the virtual address that maps to the given physical address. This provides you with a way to use SDA commands that do not have a /PHYSICAL qualifier when only the physical address of a memory location is known.

Examples

1. SDA> SHOW ADDRESS 80000000

   FFFFFFFF.80000000 is an S0/S1 address

   Mapped by Level-3 PTE at: FFFFFFFD.FFE00000
   Mapped by Level-2 PTE at: FFFFFFFD.FF7FF800
   Mapped by Level-1 PTE at: FFFFFFFD.FF7FC000
   Mapped by Selfmap PTE at: FFFFFFFD.FF7FDFF0
   Also mapped in SPT window at: FFFFFFFF.FFDF0000

   The SHOW ADDRESS command in this example shows where the address 80000000 is mapped at different page table entry levels.

2. SDA> SHOW ADDRESS 0

   00000000.00000000 is a P0 address

   Mapped by Level-3 PTE at: FFFFFFFF.C0000000
   Mapped by Level-2 PTE at: FFFFFFFF.FF000000
   Mapped by Level-1 PTE at: FFFFFFFF.FF7FC000
   Mapped by Selfmap PTE at: FFFFFFFF.FF7FDFF0

   The SHOW ADDRESS command in this example shows where the address 0 is mapped at different page table entry levels.
3. SDA> SHOW ADDRESS FFFFFFFD.FF000000

    FFFFFFFD.FF000000 is the address of a process-private Level-2 PTE
    Mapped by Level-1 PTE at: FFFFFFFD.FF7FC000
    Mapped by Selfmap PTE at: FFFFFFFD.FF7FDFF0

    Range mapped at level 2: FFFFFFFC.00000000 to FFFFFFFC.0001FFFF (1 page)
    Range mapped at level 3: 00000000.00000000 to 00000000.007FFFFF (1024 pages)

    The SHOW ADDRESS command in this example shows where the address
    FFFFFFFD.FF7FC000 is mapped at page table entry and the range mapped
    by the PTE at this address.

4. SDA> SHOW ADDRESS/PHYSICAL 0

    Physical address 00000000.00000000 is mapped to system-space address FFFFFFFF.828FC000

    The SHOW ADDRESS command in this example shows physical address
    00000000.00000000 mapped to system-space address FFFFFFFF.828FC000.

5. SDA> SHOW ADDRESS/PHYSICAL 029A6000

    Physical address 00000000.029A6000 is mapped to process-space address 00000000.00030000
    (process index 0024)

    The SHOW ADDRESS command in this example shows physical address
    00000000.029A6000 mapped to process-space address 00000000.00030000
    (process index 0024).
SHOW BUGCHECK

Displays the value, name, and text associated with one or all bugcheck codes.

Format

SHOW BUGCHECK {/ALL (d)|name|number}

Parameters

name
Displays the value, name, and text of the named bugcheck code.

number
Displays the value, name, and text of the requested bugcheck code.

The parameters name and number, and the qualifier /ALL, are all mutually exclusive.

Qualifier

/ALL
Displays complete list of all the bugcheck codes giving their value, name, and text. It is the default.

Description

The SHOW BUGCHECK command displays the value, name, and text associated with bugcheck codes.

Examples

1. SDA> show bugcheck 100
   0100 DIRENTRY ACP failed to find same directory entry
   
   The SHOW BUGCHECK command in this example shows the requested bugcheck by number.

2. SDA> show bugcheck decnet
   08D0 DECNET DECnet detected a fatal error
   
   The SHOW BUGCHECK command in this example shows the requested bugcheck by name.

3. SDA> show bugcheck
   BUGCHECK codes and texts
   ------------------------
   0008 ACPMBFAIL ACP failure to read mailbox
   0010 ACPVAFAIL ACP failure to return virtual address space
   0018 ALCPHD Allocate process header error
   0020 ALCSMBCLR ACP tried to allocate space already allocated
   .
   .
   .
   
   The SHOW BUGCHECK command in this example shows the requested bugcheck by displaying all codes.
SHOW CALL_FRAME

Displays the locations and contents of the quadwords representing a procedure call frame.

Format

SHOW CALL_FRAME {starting-address} [/NEXT_FP]

Parameter

starting-address
Expression representing the starting address of the procedure call frame to be displayed. The default starting-address is the contents of the FP register of the SDA current process.

Qualifier

/NEXT_FP
Displays the procedure call frame starting at the address stored in the FP longword of the last call frame displayed by this command. You must have issued a SHOW CALL_FRAME command previously in the current SDA session in order to use the /NEXT_FP qualifier to the command.

Description

Whenever a procedure is called, information is stored on the stack of the calling routine in the form of a procedure call frame. The SHOW CALL_FRAME command displays the locations and contents of the call frame. The starting address of the call frame is determined from the specified starting address, the /NEXT_FP qualifier, or the address contained in the SDA current process FP register (the default action).

When using the SHOW CALL_FRAME/NEXT_FP command to follow a chain of call frames, SDA signals the end of the chain by this message:

%SDA-E-NOTINPHYS, 00000000.00000000 : not in physical memory

This message indicates that the saved FP in the previous call frame has a zero value.

Example

SDA> SHOW CALL_FRAME
Call Frame Information
---------------------------
Stack Frame Procedure Descriptor
Flags: Base Register = FP, No Jacket, Native
 Procedure Entry: FFFFFFFF.837E9F10 EXCEPTION_PRO+01F10
 Return address on stack = FFFFFFFF.837E8A1C EXE$CONT_SIGNAL_C+0019C

4–64 SDA Commands
SDA Commands

SHOW CALL_FRAME

Registers saved on stack
------------------------
7FF99F98 FFFFFFF.FFFFFFFFB Saved R2
7FF99FA0 FFFFFFF.F8042AE0A Saved R3 EXCEPTION_NPRW+040A0
7FF99FA8 00000000.00000002 Saved R5
7FF99FB0 FFFFFFF.F804344A0 Saved R13 SCH$CLREF+00188
7FF99FB8 00000000.7FF9FC00 Saved R29

SDA> SHOW CALL_FRAME/NEXT_FP
Call Frame Information
------------------------

Stack Frame Procedure Descriptor
Flags: Base Register = FP, No Jacket, Native
Procedure Entry: FFFFFFF.800FA388 RMS_NPRO+04388
Return address on stack = FFFFFFF.80040BFC EXCEPTION_NPRO+00BFC

Registers saved on stack
------------------------
7FF99F60 FFFFFFF.FFFFFFFFD Saved R2
7FF99F68 FFFFFFF.80425BA0 Saved R3 EXCEPTION_NPRW+03DA0
7FF99F70 FFFFFFF.80422020 Saved R4 EXCEPTION_NPRW+00220
7FF99F78 00000000.00000000 Saved R5
7FF99F80 FFFFFFF.835C24A8 Saved R6 RMS_PRO+004A8
7FF99F88 00000000.7FF9FD88 Saved R7
7FF99F90 00000000.7FF9FD90 Saved R8
7FF99F98 00000000.7FF9F9F0 Saved R9
7FF99FA0 00000000.7FF9FE78 Saved R10
7FF99FA8 00000000.7FF9FEBC Saved R11
7FF99FB0 FFFFFFF.837626E0 Saved R13 EXE$OPEN_MESSAGE+00088
7FF99FB8 00000000.7FF9F7D0 Saved R29

SDA> SHOW CALL_FRAME/NEXT_FP
Call Frame Information
------------------------

Stack Frame Procedure Descriptor
Flags: Base Register = FP, No Jacket, Native
Procedure Entry: FFFFFFF.835C2438 RMS_PRO+00438
Return address on stack = FFFFFFF.83766020 EXE$OPEN_MESSAGE_C+00740

Registers saved on stack
------------------------
7FF9FD88 00000000.7FF9F9EA4 Saved R2
7FF9FD90 00000000.7FF9FE00 Saved R3
7FF9FD98 00000000.7FFA0050 Saved R29

The SHOW CALL_FRAME commands in this SDA session follow a chain of call frames from that specified in the FP of the SDA current process.
SHOW CLUSTER

Displays connection manager and system communications services (SCS) information for all nodes in a cluster.

Format

SHOW CLUSTER  {[(/ADDRESS=n|/CSID=csid|/NODE=name)]|/SCS}

Parameters

None.

Qualifiers

/ADDRESS=n
Displays only the OpenVMS Cluster system information for a specific OpenVMS Cluster member node, given the address of the cluster system block (CSB) for the node. This is mutually exclusive with the /CSID=csid and /NODE=name qualifiers.

/CSID=csid
Displays only the OpenVMS Cluster system information for a specific OpenVMS Cluster member node. The value csid is the cluster system identification number (CSID) of the node to be displayed. You can find the CSID for a specific node in a cluster by examining the CSB list display of the SHOW CLUSTER command. Other SDA displays refer to a system's CSID. For instance, the SHOW LOCK command indicates where a lock is mastered or held by CSID. This is mutually exclusive with the /ADDRESS=n and /NODE=name qualifiers.

/NODE=name
Displays only the OpenVMS Cluster system information for a specific OpenVMS Cluster member node, given its SCS node name. This is mutually exclusive with the /ADDRESS=n and /CSID=csid qualifiers.

/SCS
Displays a view of the cluster as seen by SCS.

Description

The SHOW CLUSTER command provides a view of the OpenVMS Cluster system from either the perspective of the connection manager (the default behavior), or from the perspective of the port driver(s) (if the /SCS qualifier is used).

OpenVMS Cluster as Seen by the Connection Manager
The SHOW CLUSTER command provides a series of displays.

The OpenVMS Cluster summary display supplies the following information:

• Number of votes required for a quorum
• Number of votes currently available
• Number of votes allocated to the quorum disk
• Status summary indicating whether or not a quorum is present
The **CSB list** displays information about the OpenVMS Cluster system blocks (CSBs) currently in operation; there is one CSB assigned to each node of the cluster. For each CSB, the **CSB list** displays the following information:

- Address of the CSB
- Name of the OpenVMS Cluster node it describes
- CSID associated with the node
- Number of votes (if any) provided by the node
- State of the CSB
- Status of the CSB

For information about the state and status of nodes, see the description of the ADD CLUSTER command of the SHOW CLUSTER utility in the OpenVMS System Management Utilities Reference Manual.

The **cluster block** display includes information recorded in the cluster block (CLUB), including a list of activated flags, a summary of quorum and vote information, and other data that applies to the cluster from the perspective of the node for which the SDA is being run.

The **cluster failover control block** display provides detailed information concerning the cluster failover control block (CLUFCB). The **cluster quorum disk control block** display provides detailed information from the cluster quorum disk control block (CLUDCB).

Subsequent displays provide information for each CSB listed previously in the **CSB list** display. Each display shows the state and flags of a CSB, as well as other specific node information. (See the ADD MEMBER command of the SHOW CLUSTER utility in the OpenVMS System Management Utilities Reference Manual for information about the flags for OpenVMS Cluster nodes.)

If any of the qualifiers /ADDRESS=nn, /CSID=csid, or /NODE=name are specified, then the SHOW CLUSTER command displays only the information from the CSB of the specified node.

**OpenVMS Cluster as Seen by the Port Driver**

The SHOW CLUSTER/SCS command provides a series of displays.

The **SCS listening process directory** lists those processes that are listening for incoming SCS connect requests. For each of these processes, this display records the following information:

- Address of its directory entry
- Connection ID
- Name
- Explanatory information, if available

The **SCS systems summary** display provides the system block (SB) address, node name, system type, system ID, and the number of connection paths for each SCS system. An **SCS system** can be a OpenVMS Cluster member, storage controller, or other such device.
Subsequent displays provide detailed information for each of the system blocks and the associated path blocks. The system block displays include the maximum message and datagram sizes, local hardware and software data, and SCS poller information. Path block displays include information that describes the connection, including remote functions and other path-related data.

Examples

1. SDA> SHOW CLUSTER
   OpenVMS Cluster data structures

   --- OpenVMS Cluster Summary ---
   Quorum Votes Quorum Disk Votes Status Summary
   ----- ----- ----------------- --------------
   2      2             1           qf_dynvote,qf_vote,quorum

   --- CSB list ---
   Address Node CSID Votes State Status
   ------- ---- ---- ----- ----- -----
   805FA780 FLAM5 00010006 0    local member,qf_same,qf_noaccess
   8062C400 ROMRDR 000100ED 1     open member,qf_same,qf_watcher,qf_active
   8062C780 VANDQ1 000100EF 0     open member,qf_same,qf_noaccess

   --- Cluster Block (CLUB) 805FA380 ---
   Flags: 16080005 cluster,qf_dynvote,init,qf_vote,qf_newvote,quorum
   Quorum/Votes 2/2 Last transaction code 02
   Quorum Disk Votes 1 Last trans. number 596
   Nodes 3 Last coordinator CSID 000100EF
   Quorum Disk $1$DIA0 Last time stamp 31-DEC-1992
   Found Node SYSID 00000000FC03
   Founding Time 3-JAN-1993 Largest trans. id 00000254
   Timer Entry Address 00000000
   Index of next CSID 0007 Figure of Merit 00000000
   Quorum Disk Ctrl Block 805FADC0 Member State Seq. Num 0203
   Watcher CSID 000100ED

   --- Cluster Failover Control Block (CLUFCB) 805FA4C0 ---
   Flags: 00000000
   Failover Step Index 00000037 CSB of Synchr. System 8062C780
   Failover Instance ID 00000254

   --- Cluster Quorum Disk Control Block (CLUDCB) 805FADC0 ---
   State : 0002 qa_rem_act
   Flags : 0100 qf_noaccess
   CSP Flags : 0000
   Iteration Counter 0 UCB address 00000000
   Activity Counter 0 TQE address 805FAE00
   Quorum file LBN 00000000 IRP address 00000000
   Watcher CSID 000100ED

   --- FLAM5 Cluster System Block (CSB) 805FA780 ---
   State: 0B local
   Flags: 070260AA member,qf_same,qf_noaccess,selected,local,status_rcvd,send_status
   Cpblty: 00000000
   SWVers: 7.0
   HName: DEC 3000 Model 400
This example illustrates the default output of the SHOW CLUSTER command.
SDA Commands
SHOW CLUSTER

2. SDA> SHOW CLUSTER/SCS

VMS cluster data structures
-------------------------------------
--- SCS Listening Process Directory ---
<table>
<thead>
<tr>
<th>Entry Address</th>
<th>Connection ID</th>
<th>Process Name</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>80C71EC0</td>
<td>74D20000</td>
<td>SCSSDIRECTORY</td>
<td>Directory Server</td>
</tr>
<tr>
<td>80C72100</td>
<td>74D20001</td>
<td>MSCP$TAPE</td>
<td>NOT PRESENT HERE</td>
</tr>
<tr>
<td>80E6940</td>
<td>74D20002</td>
<td>MSCP$DISK</td>
<td></td>
</tr>
<tr>
<td>80E23B40</td>
<td>74D20003</td>
<td>VMSSSDA_AXP</td>
<td>Remote SDA</td>
</tr>
<tr>
<td>80E23B40</td>
<td>74D20003</td>
<td>VMSSSDA_AXP</td>
<td>Remote SDA</td>
</tr>
<tr>
<td>80E25540</td>
<td>74D20004</td>
<td>VMSSVAXcluster</td>
<td></td>
</tr>
<tr>
<td>80E29F80</td>
<td>74D20005</td>
<td>SCA$TRANSPORT</td>
<td></td>
</tr>
<tr>
<td>813020C0</td>
<td>74D20053</td>
<td>PATHWORKScluster</td>
<td>TurboServer</td>
</tr>
</tbody>
</table>

--- SCS Systems Summary ---
<table>
<thead>
<tr>
<th>SB Address</th>
<th>Node</th>
<th>Type</th>
<th>System ID</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>8493BC00</td>
<td>ARUSHA</td>
<td>VMS</td>
<td>000000004CA1</td>
<td>2</td>
</tr>
<tr>
<td>80E23E00</td>
<td>HSJ201</td>
<td>HSJ</td>
<td>4200101A1B20</td>
<td>1</td>
</tr>
<tr>
<td>80E33F00</td>
<td>ORNOT</td>
<td>VMS</td>
<td>000000004CA7</td>
<td>2</td>
</tr>
<tr>
<td>80E33F00</td>
<td>LOADQ</td>
<td>VMS</td>
<td>000000004C31</td>
<td>2</td>
</tr>
<tr>
<td>80E47CC0</td>
<td>HSJ300</td>
<td>HSJ</td>
<td>420010051D20</td>
<td>1</td>
</tr>
<tr>
<td>80E47CC0</td>
<td>HSJ101</td>
<td>HSJ</td>
<td>420010081720</td>
<td>1</td>
</tr>
<tr>
<td>80E47CC0</td>
<td>HSJ100</td>
<td>HSJ</td>
<td>4200100B1520</td>
<td>1</td>
</tr>
<tr>
<td>80E47CC0</td>
<td>HSJ600</td>
<td>HSJ</td>
<td>42001007920</td>
<td>1</td>
</tr>
<tr>
<td>80E49180</td>
<td>HSJ401</td>
<td>HSJ</td>
<td>4200100320</td>
<td>1</td>
</tr>
<tr>
<td>80E47CD0</td>
<td>HSJ301</td>
<td>HSJ</td>
<td>420010091P20</td>
<td>1</td>
</tr>
<tr>
<td>80E47CD0</td>
<td>HSJ601</td>
<td>HSJ</td>
<td>4200100A0B20</td>
<td>1</td>
</tr>
<tr>
<td>80E4F500</td>
<td>HSJ400</td>
<td>HSJ</td>
<td>4200100C120</td>
<td>1</td>
</tr>
<tr>
<td>80E5F800</td>
<td>CHOBES</td>
<td>VMS</td>
<td>000000004CD6</td>
<td>2</td>
</tr>
<tr>
<td>80E5F800</td>
<td>ETOSHA</td>
<td>VMS</td>
<td>000000004CF3</td>
<td>2</td>
</tr>
<tr>
<td>80E5FC00</td>
<td>VMS</td>
<td>VMS</td>
<td>000000004C7A</td>
<td>2</td>
</tr>
<tr>
<td>80E4F800</td>
<td>HSJ501</td>
<td>HSJ</td>
<td>4200101C0720</td>
<td>1</td>
</tr>
<tr>
<td>80E4F800</td>
<td>HSJ200</td>
<td>HSJ</td>
<td>420010191920</td>
<td>1</td>
</tr>
<tr>
<td>80E4F800</td>
<td>HSJ500</td>
<td>HSJ</td>
<td>4200101B0520</td>
<td>1</td>
</tr>
<tr>
<td>80E5F800</td>
<td>IPL31</td>
<td>VMS</td>
<td>000000004F52</td>
<td>2</td>
</tr>
<tr>
<td>80E5F800</td>
<td>ZAPNOT</td>
<td>VMS</td>
<td>000000004CBB</td>
<td>2</td>
</tr>
<tr>
<td>80E6F800</td>
<td>ALTOS</td>
<td>VMS</td>
<td>000000004DOF</td>
<td>2</td>
</tr>
<tr>
<td>80E6F800</td>
<td>TSAND</td>
<td>VMS</td>
<td>000000004CFE</td>
<td>2</td>
</tr>
<tr>
<td>80E6F800</td>
<td>SLYTHE</td>
<td>VMS</td>
<td>000000004DD1</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>AZSUN</td>
<td>VMS</td>
<td>000000004D56</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>CALSUN</td>
<td>VMS</td>
<td>000000004EA4</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>4X4TRK</td>
<td>VMS</td>
<td>000000004F26</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>GNS</td>
<td>VMS</td>
<td>000000004FC2B</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>IXIVIV</td>
<td>VMS</td>
<td>000000004E56</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>CLAIR</td>
<td>VMS</td>
<td>000000004CDF</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>INT4</td>
<td>VMS</td>
<td>000000004D70</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>SCOPE</td>
<td>VMS</td>
<td>000000004C87</td>
<td>1</td>
</tr>
<tr>
<td>80E6F800</td>
<td>MOCKUP</td>
<td>VMS</td>
<td>000000004CDF5</td>
<td>1</td>
</tr>
</tbody>
</table>

--- ARUSA System Block (SB) 8493BC00 ---

System ID: 000000004CA1  Local software type: VMS
Max message size: 216  Local software vers.: V7.2
Max datagram size: 576  Local software incarn.: DF4AC300
Local hardware type: ALPH 009F7570
Local hardware vers. 000000000003  SCS poller timeout: 5AD3
Local hardware vers. 040000000000  SCS poller enable mask: 27
Status: 00000000
This example illustrates the output of the SHOW CLUSTER /SCS command.
SHOW CONNECTIONS

Displays information about all active connections between System Communications Services (SCS) processes or a single connection.

Format

SHOW CONNECTIONS  [/[ADDRESS=cdt-address | /NODE=name | /SYSAP=name]]

Parameters

None.

Qualifiers

/ADDRESS=cdt-address
Displays information contained in the connection descriptor table (CDT) for a specific connection. You can find the cdt-address for any active connection on the system in the CDT summary page display of the SHOW CONNECTIONS command. In addition, CDT addresses are stored in many individual data structures related to SCS connections. These data structures include class driver request packets (CDRP)s and unit control blocks (UCB)s for class drivers that use SCS, and cluster system blocks (CSB)s for the connection manager.

/NODE=name
Displays all CDTs associated with the specified remote SCS node name.

/SYSAP=name
Displays all CDTs associated with the specified local SYSAP.

Description

The SHOW CONNECTIONS command provides a series of displays.

The CDT summary page lists information regarding each connection on the local system, including the following:

• CDT address
• Name of the local process with which the CDT is associated
• Connection ID
• Current state
• Name of the remote node (if any) to which it is currently connected

The CDT summary page concludes with a count of CDTs that are free and available to the system.

SHOW CONNECTIONS next displays a page of detailed information for each active CDT listed previously.
**Example**

**SDA> SHOW CONNECTIONS**

--- CDT Summary Page ---

<table>
<thead>
<tr>
<th>CDT Address</th>
<th>Local Process</th>
<th>Connection ID</th>
<th>State</th>
<th>Remote Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>805E7ED0</td>
<td>SCSSDIRECTORY</td>
<td>FF120000</td>
<td>listen</td>
<td></td>
</tr>
<tr>
<td>805E8030</td>
<td>MSCP$TAPE</td>
<td>FF120001</td>
<td>listen</td>
<td></td>
</tr>
<tr>
<td>805E8190</td>
<td>VMSSVMSccluster</td>
<td>FF120002</td>
<td>listen</td>
<td></td>
</tr>
<tr>
<td>805E82F0</td>
<td>MSCP$DISK</td>
<td>FF120003</td>
<td>listen</td>
<td></td>
</tr>
<tr>
<td>805E8450</td>
<td>SCA$TRANSPORT</td>
<td>FF120004</td>
<td>listen</td>
<td></td>
</tr>
<tr>
<td>805E8580</td>
<td>MSCP$DISK</td>
<td>FF150005</td>
<td>open</td>
<td>VANDQ1</td>
</tr>
<tr>
<td>805E8710</td>
<td>VMSSVMSccluster</td>
<td>FF120006</td>
<td>open</td>
<td>VANDQ1</td>
</tr>
<tr>
<td>805E8870</td>
<td>VMSSVMSccluster</td>
<td>FF120007</td>
<td>open</td>
<td>ROMRDR</td>
</tr>
<tr>
<td>805E8990</td>
<td>MSCP$DISK</td>
<td>FF120008</td>
<td>open</td>
<td>ROMRDR</td>
</tr>
<tr>
<td>805E8C90</td>
<td>VMSSDISK_CL_DRVR</td>
<td>FF12000A</td>
<td>open</td>
<td>ROMRDR</td>
</tr>
<tr>
<td>805E8DF0</td>
<td>VMSSDISK_CL_DRVR</td>
<td>FF12000B</td>
<td>open</td>
<td>VANDQ1</td>
</tr>
<tr>
<td>805E8F50</td>
<td>VMSSTAPE_CL_DRVR</td>
<td>FF12000C</td>
<td>open</td>
<td>VANDQ1</td>
</tr>
</tbody>
</table>

Number of free CDT's: 188

--- Connection Descriptor Table (CDT) 80C44850 ---

State: 0001 listen  Local Process: MSCP$TAPE
Blocked State: 0000

<table>
<thead>
<tr>
<th>Local Con. ID</th>
<th>Datagrams sent</th>
<th>Remote Con. ID</th>
<th>Datagrams rcvd</th>
<th>Send Credit Q.</th>
<th>Send Credit</th>
<th>Min. Rec. Credit</th>
<th>Message Recvs</th>
<th>Pend Rec. Credit</th>
<th>Message Sends NoFP</th>
<th>Initial Rec. Credit</th>
<th>Message Recvs NoFP</th>
<th>Rem. Sta.</th>
<th>Send Data Init.</th>
<th>Req Data Init.</th>
<th>Queued for BDLT</th>
<th>Bytes Sent</th>
<th>Fast Recvmsg</th>
<th>Send Data Init.</th>
<th>Req Data Init.</th>
<th>Queued Send Credit</th>
<th>Bytes rcvd</th>
<th>Change Affinity</th>
<th>Total bytes map</th>
</tr>
</thead>
<tbody>
<tr>
<td>899F0003</td>
<td>0</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

--- Connection Descriptor Table (CDT) 805E8030 ---

State: 0001 listen  Local Process: MSCP$TAPE
Blocked State: 0000

<table>
<thead>
<tr>
<th>Local Con. ID</th>
<th>Datagrams sent</th>
<th>Remote Con. ID</th>
<th>Datagrams rcvd</th>
<th>Send Credit Q.</th>
<th>Send Credit</th>
<th>Min. Rec. Credit</th>
<th>Message Recvs</th>
<th>Pend Rec. Credit</th>
<th>Message Sends NoFP</th>
<th>Initial Rec. Credit</th>
<th>Message Recvs NoFP</th>
<th>Rem. Sta.</th>
<th>Send Data Init.</th>
<th>Req Data Init.</th>
<th>Queued for BDLT</th>
<th>Bytes Sent</th>
<th>Fast Recvmsg</th>
<th>Send Data Init.</th>
<th>Req Data Init.</th>
<th>Queued Send Credit</th>
<th>Bytes rcvd</th>
<th>Change Affinity</th>
<th>Total bytes map</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF120001</td>
<td>0</td>
<td>00000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

This example shows the default output of the SHOW CONNECTIONS command.
SHOW CPU

When analyzing a dump, displays information about the state of a CPU at the time of the system failure.

-------------------- Note ---------------------
SHOW CPU is only valid when you are analyzing a crash dump. It is not a valid command when you are analyzing the running system, because all the CPU-specific information may not be available.

Format

SHOW CPU [cpu-id]

Parameter

cpu-id
Numeric value from 00 to 1F_{16} indicating the identity of the CPU for which context information is to be displayed. If you specify a value outside this range, or you specify the cpu-id of a CPU that was not active at the time of the system failure, SDA displays the following message:

%SDA-E-CPUNOTVLD, CPU not booted or CPU number out of range

If you use the cpu-id parameter, the SHOW CPU command performs an implicit SET CPU command, making the CPU indicated by cpu-id the current CPU for subsequent SDA commands. (See the description of the SET CPU command and Chapter 2, Section 2.5 for information on how this can affect the CPU context—and process context—in which SDA commands execute.)

Qualifiers

None.

Description

The SHOW CPU command displays system failure information about the CPU specified by cpu-id or, by default, the SDA current CPU, as defined in Chapter 2, Section 2.5. You cannot use the SHOW CPU command when examining the running system with SDA.

The SHOW CPU command produces several displays. First, there is a brief description of the system failure and its environment that includes the following:

• Reason for the bugcheck.
• Name of the currently executing process. If no process has been scheduled on this CPU, SDA displays the following message:

Process currently executing: no processes currently scheduled on the processor

• File specification of the image executing within the current process (if there is a current process).
• Interrupt priority level (IPL) of the CPU at the time of the system failure.
• The CPU database address.
The CPU’s capability set.

Next, the **general registers** display shows the contents of the CPU’s integer registers (R0 to R30), and the AI, RA, PV, FP, PC, and PS at the time of the system failure.

The **processor registers** display consists of the following parts:

- Common processor registers
- Processor-specific registers
- Stack pointers

The first part of the processor registers display includes registers common to all Alpha processors, which are used by the operating system to maintain the current process virtual address space, system space, or other system functions. This part of the display includes the following registers:

- Hardware privileged context block base register (PCBB)
- System control block base register (SCBB)
- Software interrupt summary register (SISR)
- Address space number register (ASN)
- AST summary register (ASTSR)
- AST enable register (ASTEN)
- Interrupt priority level register (IPL)
- Processor priority level register (PRBR)
- Page table base register (PTBR)
- Virtual page table base register (VPTB)
- Floating point control register (FPCR)
- Machine check error summary register (MCES)

The last part of the display includes the four stack pointers: the pointers of the kernel, executive, supervisor, and user stacks (KSP, ESP, SSP, and USP, respectively).

The SHOW CPU command concludes with a listing of the spinlocks, if any, owned by the CPU at the time of the system failure, reproducing some of the information given by the SHOW SPINLOCKS command. The spinlock display includes the following information:

- Name of the spinlock.
- Address of the spinlock data structure (SPL).
- The owning CPU’s CPU ID.
- IPL of the spinlock.
- Indication of the depth of this CPU’s ownership of the spinlock. A number greater than 1 indicates that this CPU has nested acquisitions of the spinlock.
- Rank of the spinlock.
- Timeout interval for spinlock acquisition (in terms of 10 milliseconds).
- Shared array (shared spinlock context block pointers)
Example

SDA> SHOW CPU 0
CPU 00 Processor crash information
----------------------------------
CPU 00 reason for Bugcheck: CPUEXIT, Shutdown requested by another CPU

Process currently executing on this CPU: None

Current IPL: 31 (decimal)

CPU database address: 81414000

CPUs Capabilities: PRIMARY, QUORUM, RUN

General registers:

R0 = FFFFFFFF.81414000 R1 = FFFFFFFF.81414000 R2 = 00000000.00000000
R3 = FFFFFFFF.810AD960 R4 = 00000000.01668E90 R5 = 00000000.00000001
R6 = 66666666.66666666 R7 = 77777777.77777777 R8 = FFFFFFFF.814FB040
R9 = 99999999.99999999 R10 = FFFFFFFF.814FB0C0 R11 = BBBBBBBB.BBBBBBBB
R12 = 55555555.55555555 R13 = FFFFFFFF.81AD960 R14 = FFFFFFFF.8141018
R15 = 00000000.00000004 R16 = 00000000.0000006AC R17 = 00000000.00000047
R18 = 00000000.00000000 R19 = 00000000.00000000 R20 = FFFFFFFF.8051A494
R21 = 00000000.00000000 R22 = 00000000.00000001 R23 = 00000000.00000010
R24 = FFFFFFFF.81414000 A1 = FFFFFFFF.81414000 R28 = 00000000.00000000
PV = 00000001.FFFFFFFF R29 = 00000000.00000000 FP = FFFFFFFF.8051A494
PC = FFFFFFFF.8009C95C PS = 18000000.00001F04

Processor Internal Registers:

ASN = 00000000.00000000 ASTSR/ASTEN = 00000000
IPL = 0000001F PCBB = 00000000.01014080 PRBR = FFFFFFFF.81414000
PTBR = 00000000.0000FFBF SCBB = 00000000.000001E8 SISR = 00000000.00000100
VPTB = FFFFFFFF.8009C95C FPCR = 00000000.00000000 MCES = 00000000.00000000
KSP = FFFFFFFF.8009C95C ESP = FFFFFFFF.8009C95C SSP = FFFFFFFF.8009C95C
USP = FFFFFFFF.8009C95C

Spinlocks currently owned by CPU 00

SCS Address 810AF300
Owner CPU ID 00000000 IPL 00000008
Ownership Depth 00000000 Rank 0000001A
Timeout Interval 002DC6C0 Share Array 00000000

This example shows the default output of the SHOW CPU command.
SHOW CRASH

Displays information about the state of the system at the time of failure. Provides a system information identifying a running system.

Format

SHOW CRASH [/CPU=n]

Parameters

None.

Qualifier

/CPU=n

Allows exception data to be displayed from CPUs other than the one considered as the crash CPU when more than one CPU crashes simultaneously.

Description

The SHOW CRASH command has two different functions, depending on whether you use it to analyze a running system or a system failure.

When used during the analysis of a running system, the SHOW CRASH command produces a display that describes the system and the version of OpenVMS Alpha that it is running. The system crash information display contains the following information:

- Name and version number of the operating system
- Major and minor IDs of the operating system
- Identity of the Alpha system, including an indication of its cluster membership
- CPU ID of the primary CPU
- Address of all CPU databases

When used during the analysis of a system failure, the SHOW CRASH command produces several displays that identify the system and describe its state at the time of the failure.

If the current CPU context for SDA is not that of the processor that signaled the bugcheck, or the CPU specified with the /CPU=n qualifier, the SHOW CRASH command first performs an implicit SET CPU command to make that processor the current CPU for SDA. (See the description of the SET CPU command and Chapter 2, Section 2.5 for a discussion of how this can affect the CPU context—and process context—in which SDA commands execute.)

The system crash information display in this context provides the following information:

- Date and time of the system failure.
- Name and version number of the operating system.
- Major and minor IDs of the operating system.
- Identity of the system.
• CPU IDs of both the primary CPU and the CPU that initiated the bugcheck. In a uniprocessor Alpha system, these IDs are identical.

• Bitmask of the active and available CPUs in the system.

• For each active processor in the system, the name of the bugcheck that caused the system failure. Generally, there will be only one significant bugcheck in the system. All other processors typically display the following as their reason for taking a bugcheck:

  CPUEXIT, Shutdown requested by another CPU

Subsequent screens of the SHOW CRASH command display information about the state of each active processor on the system at the time of the system failure. The information in these screens is identical to that produced by the SHOW CPU command, including the general-purpose registers, processor-specific registers, stack pointers, and records of spinlock ownership. The first such screen presents information about the processor that caused the failure; others follow according to the numeric order of their CPU IDs.

Examples

1. SDA> SHOW CRASH

   System crash information
   ------------------------
   Time of system crash: 1-JAN-2001 00:00:00.00
   Version of system: OpenVMS (TM) Alpha Operating System, Version X901-SSB
   System Version Major ID/Minor ID: 3/0
   VMScluster node: VMSTS6, a
   Crash CPU ID/Primary CPU ID: 00/00
   Bitmask of CPUs active/available: 00000001/00000001
   CPU bugcheck codes:
      CPU 00 -- INVEXCEPTN, Exception while above ASTDEL

   System State at Time of Exception
   -------------------------------
   Exception Frame:
   ----------------
   R2 = FFFFFFFF.810416C0  SCSS$GA_LOCALSB+005C0
   R3 = FFFFFFFF.81007E60  EXE$GPL_HWRPB_L
   R4 = FFFFFFFF.850AEB80
   R5 = FFFFFFFF.81041330  SCSS$GA_LOCALSB+00230
   R6 = FFFFFFFF.810386E8  CON$INITINLINE
   R7 = FFFFFFFF.81041330  SCSS$GA_LOCALSB+00230
   PC = FFFFFFFF.803EF81C  SYS$TTRIVFER+0F81C
   PS = 30000000.00001F04
   FFFFFFFF.803EF80C: STL R24,#X0060(R5)
   FFFFFFFF.803EF810: LDL R28,#X0138(R5)
   FFFFFFFF.803EF814: BIC R28,R27,R28
   FFFFFFFF.803EF818: 00000138
   PC => FFFFFFFF.803EF81C: HALT
   FFFFFFFF.803EF820: HALT
   FFFFFFFF.803EF824: BR R31,#XFF0000
   FFFFFFFF.803EF828: LDL R24,#X0138(R5)
   FFFFFFFF.803EF82C: BIC R24,#X40,R24
   PS =>
   MBZ SPAL MBZ IPL VMM MBZ CURMOD INT PRVMD de 0 30 000000000001F 0 0 KERN 1 KNR
Signal Array
------------
Length = 00000003
Type = 0000043C
Arg = FFFFFFFF.803EF81C SYS$TIDRIVER+0F81C
Arg = 30000000.00001F04
%SYSTEM-F-OPCDEC, opcode reserved to Digital fault at PC=FFFFFFFE803EF81C, PS=00001F04

Saved Scratch Registers in Mechanism Array
------------------------------------------
R0 = 00000000.00000000 R1 = FFFFFFFF.811998B8 R16 = 00000000.00001000
R17 = FFFFFFFF.811991F0 R18 = 00000000.00000010 R19 = FFFFFFFF.810194F0
R20 = 00000000.00000000 R21 = 00000000.00000000 R22 = 00000000.00000000
R23 = 00000000.00004000 R24 = 00000000.00001000 R25 = 00000000.00000000
R26 = FFFFFFFF.81041474 R27 = 00000000.00004000 R28 = 00000000.00001000

This long display reflects the output of the SHOW CRASH command within the analysis of a system failure.

2. SDA> SHOW CRASH
System crash information
------------------------
Time of system crash: 12-OCT-2000 11:27:58.02
Version of system: OpenVMS (TM) Alpha Operating System, Version X74B-FT2
System Version Major ID/Minor ID: 3/0
System type: DEC 3000 Model 400
Crash CPU ID/Primary CPU ID: 00/00
Bitmask of CPUs active/available: 00000001/00000001
CPU bugcheck codes:
  CPU 00 -- PGFIPLHI, Pagefault with IPL too high
System State at Time of Page Fault:
-----------------------------------
Page fault for address 00000000.00046000 occurred at IPL: 8
Memory management flags: 00000000.00000001 (instruction fetch)
Exception Frame:
----------------
R2 = 00000000.0000003
R3 = FFFFFFFF.810B9280 EXCEPTION_MON+39C80
R4 = FFFFFFFF.81564540 PCB
R5 = 00000000.00000088
R6 = 00000000.00045B80
R7 = 00000000.7FFA1FC0
PC = 00000000.00046000
PS = 20000000.00000803
This display reflects the output of a SHOW CRASH command within the analysis of a PGF1PLHI bugcheck.
SHOW DEVICE

Displays a list of all devices in the system and their associated data structures, or displays the data structures associated with a given device or devices.

Format

SHOW DEVICE [device-name | /ADDRESS=ucb-address | /CDT=cdt_address | /CHANNELS | /HOMEPAGE | /PDT | /UCB=ucb-address]

Parameter

device-name
Device or devices for which data structures are to be displayed. There are several uses of the device-name parameter.

<table>
<thead>
<tr>
<th>To Display the Structures For . . .</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>All devices in the system</td>
<td>Do not specify a device-name (for example, SHOW DEVICE).</td>
</tr>
<tr>
<td>A single device</td>
<td>Specify an entire device-name (for example, SHOW DEVICE VTA20).</td>
</tr>
<tr>
<td>All devices of a certain type on a single controller</td>
<td>Specify only the device type and controller designation (for example, SHOW DEVICE RTA or SHOW DEVICE RTB).</td>
</tr>
<tr>
<td>All devices of a certain type on any controller</td>
<td>Specify only the device type (for example, SHOW DEVICE RT).</td>
</tr>
<tr>
<td>All devices whose names begin with a certain character or character string</td>
<td>Specify the character or character string (for example, SHOW DEVICE D).</td>
</tr>
<tr>
<td>All devices on a single node or HSC</td>
<td>Specify only the node name or HSC name (for example, SHOW DEVICE GREEN$).</td>
</tr>
<tr>
<td>All devices with a certain allocation class</td>
<td>Specify the allocation class including leading and trailing $, for example, SHOW DEVICE $63$.</td>
</tr>
</tbody>
</table>

Qualifiers

/ADDRESS=ucb-address
Indicates the device for which data structure information is to be displayed by the address of its unit control block (UCB). The /ADDRESS qualifier is an alternate method of supplying a device name to the SHOW DEVICE command. If both the device-name parameter and the /ADDRESS qualifier appear in a single SHOW DEVICE command, SDA responds only to the parameter or qualifier that appears first.

/CDT=cdt_address
Identifies the device by the address of its Connector Descriptor Table (CDT). This applies to cluster port devices only.

/CHANNELS
Displays information on active Memory Channel channel blocks. This qualifier is ignored for devices other than memory channel.
SHOW DEVICE

/HOMEPAGE
Displays fields from the Memory Channel Home Page. This qualifier is ignored for devices other than memory channel.

/PDT
Displays the Memory Channel Port Descriptor Table. This qualifier is ignored for devices other than memory channel.

/UCB=ucb-address
This is a synonym for /ADDRESS=ucb-address as described above.

Description

The SHOW DEVICE command produces several displays taken from system data structures that describe the devices in the system configuration.

If you use the SHOW DEVICE command to display information for more than one device or one or more controllers, it initially produces the DDB (device data block) list to provide a brief summary of the devices for which it renders information in subsequent screens.

Information in the DDB list appears in five columns, the contents of which are as follows:

• Address of the device data block (DDB)
• Controller name
• Name of the ancillary control process (ACP) associated with the device
• Name of the device driver
• Address of the driver prologue table (DPT)

The SHOW DEVICE command then produces a display of information pertinent to the device controller. This display includes information gathered from the following structures:

• Device data block (DDB)
• Primary channel request block (CRB)
• Interrupt dispatch block (IDB)
• Driver dispatch table (DDT)

If the controller is an HSC controller, SHOW DEVICE also displays information from its system block (SB) and each path block (PB).

Many of these structures contain pointers to other structures and driver routines. Most notably, the DDT display points to various routines located within driver code, such as the start I/O routine, unit initialization routine, and cancel I/O routine.

For each device unit subject to the SHOW DEVICE command, SDA displays information taken from its unit control block, including a list of all I/O request packets (IRPs) in its I/O request queue. For certain mass storage devices, SHOW DEVICE also displays information from the primary class driver data block (CDDB), the volume control block (VCB), and the ACP queue block (AQB). For units that are part of a shadow set, SDA displays a summary of shadow set membership.
As it displays information for a given device unit, SHOW DEVICE defines the following symbols as appropriate:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCB</td>
<td>Address of unit control block</td>
</tr>
<tr>
<td>SB</td>
<td>Address of system block</td>
</tr>
<tr>
<td>ORB</td>
<td>Address of object rights block</td>
</tr>
<tr>
<td>DDB</td>
<td>Address of device data block</td>
</tr>
<tr>
<td>DDT</td>
<td>Address of driver dispatch table</td>
</tr>
<tr>
<td>CRB</td>
<td>Address of channel request block</td>
</tr>
<tr>
<td>AMB</td>
<td>Associated mailbox UCB pointer</td>
</tr>
<tr>
<td>IRP</td>
<td>Address of I/O request packet</td>
</tr>
<tr>
<td>2P_UCB</td>
<td>Address of alternate UCB for dual-pathed device</td>
</tr>
<tr>
<td>LNM</td>
<td>Address of logical name block for mailbox</td>
</tr>
<tr>
<td>PDT</td>
<td>Address of port descriptor table</td>
</tr>
<tr>
<td>CDDB</td>
<td>Address of class driver descriptor block for MSCP served device</td>
</tr>
<tr>
<td>2P_CDDB</td>
<td>Address of alternate CDDB for MSCP served device</td>
</tr>
<tr>
<td>RWAITCNT</td>
<td>Resource wait count for MSCP served device</td>
</tr>
<tr>
<td>VCB</td>
<td>Address of volume control block for mounted device</td>
</tr>
</tbody>
</table>

If you are examining a driver-related system failure, you may find it helpful to issue a SHOW STACK command after the appropriate SHOW DEVICE command, to examine the stack for any of these symbols. Note, however, that although the SHOW DEVICE command defines those symbols relevant to the last device unit it has displayed, and redefines symbols relevant to any subsequently displayed device unit, it does not undefine symbols. (For instance, SHOW DEVICE DUA0 defines the symbol PDT, but SHOW DEVICE MBA0 does not undefine it, even though the PDT structure is not associated with a mailbox device.) To maintain the accuracy of such symbols that appear in the stack listing, use the DEFINE command to modify the symbol name. For example:

SDA> DEFINE DUA0_PDT PDT
SDA> DEFINE MBA0_UCB UCB

See the descriptions of the READ and FORMAT commands for additional information on defining and examining the contents of device data structures.

Examples

1. SDA> SHOW DEVICE/ADDRESS=8041E540
   OPA0       VT300_Series       UCB address 8041E540
   Device status: 00000010 online
   Characteristics: 0C040007 rec,ccl,trm,avl,idv,odv
   00000200 nmm

SDA Commands 4–83
SDA Commands
SHOW DEVICE

SDA Commands
SHOW DEVICE

Owner UIC [000001,000004]  Operation count 160  ORB address 8041E4E8
PID 00010008  Error count 0  DDB address 8041E3F8
Class/Type 42/70  Reference count 2  DDT address 8041E438
Def. buf. size 80  BOFF 00000001  CRB address 8041E740
DEVDEND 180093A0  Byte count 0000012C  I/O wait queue 8041E5AC
DEVDEND2 FB101000  SVAPTE 80537B80
DEVDEND3 00000000  DEVSTS 00000001
FLCK index 3A
DDL address 8041E880

*** I/O request queue is empty ***

This example reproduces the SHOW DEVICE display for a single device unit, OPA0. Whereas this display lists information from the UCB for OPA0, including some addresses of key data structures and a list of pending I/O requests for the unit, it does not display information about the controller or its device driver. To display the latter information, specify the device-name as OPA (for example, SHOW DEVICE OPA).

2. SDA> SHOW DEVICE DU
I/O data structures
-------------------

DDB list
--------

<table>
<thead>
<tr>
<th>Address</th>
<th>Controller</th>
<th>ACP</th>
<th>Driver</th>
<th>DPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>80DDB3C0</td>
<td>BLUE$DUA</td>
<td>F11XQP</td>
<td>SYS$DΚDRIVER</td>
<td>807735B0</td>
</tr>
<tr>
<td>8000B2B8</td>
<td>RED$DUA</td>
<td>F11XQP</td>
<td>SYS$DΚDRIVER</td>
<td>807735B0</td>
</tr>
<tr>
<td>80D08BA0</td>
<td>BIGTOP$DUA</td>
<td>F11XQP</td>
<td>SYS$DΚDRIVER</td>
<td>807735B0</td>
</tr>
<tr>
<td>80D08AE0</td>
<td>TIMEIN$DUA</td>
<td>F11XQP</td>
<td>SYS$DΚDRIVER</td>
<td>807735B0</td>
</tr>
</tbody>
</table>

Press RETURN for more.

This excerpt from the output of the SHOW DEVICE DU command illustrates the format of the DDB list. In this case, the DDB list concerns itself with those devices whose device type begins with DU. It displays devices of these types attached to various HSCs (RED$ and BLUE$) and systems in a cluster (BIGTOP$ and TIMEIN$).
SHOW DUMP

Displays formatted information from the header, error log buffers, logical memory blocks (LMBs), memory map, compression data, and a summary of the dump. Also displays hexadecimal information of individual blocks.

Format

SHOW DUMP {/ALL | /BLOCK[=m{: | ;}n]} | [/COMPRESSION_MAP[=m{: | ;}n]] | /ERROR_LOGS | /HEADER | /LMB=[{ALL | n}] | /SUMMARY | /MEMORY_MAP}

Parameters

None.

Qualifiers

/ALL
Displays the equivalent to specifying all the /SUMMARY, /HEADER, /ERROR_LOGS, /COMPRESSION_MAP, /LMB=ALL, and /MEMORY_MAP qualifiers.

/BLOCK[=m{: | ;}n]
Displays a hexadecimal dump of one or more blocks. You can specify ranges by using the following syntax:

- no value: Displays next block
- m: Displays single block
- m:n: Displays a range of blocks from m to n, inclusive
- m;n: Displays a range of blocks starting at m and continuing for n blocks

/COMPRESSION_MAP[=m{: | ;}n]
In a compressed dump, displays details of the compression data. You can specify levels of detail by using the following syntax:

- no value: Displays a summary of all compression map blocks
- m: Displays contents of a single compression map block
- m:n: Displays details of single compression map entry

/ERROR_LOGS
Displays a summary of the error log buffers.

/HEADER
Displays the formatted contents of the dump header.

/LMB=[{ALL | n}]
In a selective dump, displays the formatted contents of logical memory block (LMB) headers and the virtual address (VA) ranges within the LMB. LMBs to be displayed can be expressed by using the following syntax:

- no value: Displays next LMB
SHOW DUMP

SHOW DUMP

n Displays LMB at block n of the dump
ALL Displays all LMBs

/MEMORY_MAP
In a full dump, displays the contents of the memory map.

/SUMMARY
Displays a summary of the dump. This is the default.

Description

The SHOW DUMP command displays information about the structure of the dump file. It displays the header, the error log buffers, and, if appropriate, the compression map, the logical memory block (LMB) headers and/or the memory map. Use this command when troubleshooting dump analysis problems.

Examples

1. SDA> SHOW DUMP/SUMMARY

Summary of dump file DKA300:[SYS0.SYSEXE]SYSDUMP.DMP;8

<table>
<thead>
<tr>
<th>Dump file section</th>
<th>VBN</th>
<th>Blocks</th>
<th>Uncomp VBN</th>
<th>Uncomp blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump header</td>
<td>000000001 00000002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error log buffers</td>
<td>000000003 00000020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compression map</td>
<td>000000023 00000010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0000 (PT space)</td>
<td>000000033 000000038 00000033 000000D2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0001 (S0/S1 space)</td>
<td>0000006B 0000621B 00000105 000095A5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0002 (S2 space)</td>
<td>00006286 000001A3 000096AA 00000352</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0003 (Page tables of key process &quot;SYSTEM&quot;)</td>
<td>00006429 00000005 000099FC 00000062</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0004 (Memory of key process &quot;SYSTEM&quot;)</td>
<td>0000642E 00000071 00009A5E 00000342</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0005 (Key global pages)</td>
<td>00007D7B 000002BA 0000CD8 00000312</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0006 (Page tables of process &quot;DTWM&quot;)</td>
<td>00008035 00000013 00000D0A 00000082</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0007 (Memory of process &quot;DTWM&quot;)</td>
<td>00008048 000013A3 0000D13C 000022E4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0008 (Remaining global pages)</td>
<td>0000C5E3 00000005 00019A44 00000062</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0009 (Page tables of process &quot;Milord_FTA1:&quot;)</td>
<td>0000C5E8 00000074 00019AA6 00000222</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMB 0008 (Remaining global pages)</td>
<td>0000C65C 00000DAC 00019CC8 00001255</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This example of the SHOW DUMP/SUMMARY command gives a summary of a selective dump.
### SDA Commands

#### SHOW DUMP

**Dump header**

<table>
<thead>
<tr>
<th>Header field</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DMPSW_FLAGS</strong></td>
<td>Flags</td>
<td>0FC1</td>
</tr>
<tr>
<td><strong>DMPSV_OLD DUMP</strong></td>
<td>Dump has been analyzed</td>
<td></td>
</tr>
<tr>
<td><strong>DMPSV_WRITECOMP</strong></td>
<td>Dump write was completed</td>
<td></td>
</tr>
<tr>
<td><strong>DMPSV_ERRLOGCOMP</strong></td>
<td>Error log buffers written</td>
<td></td>
</tr>
<tr>
<td><strong>DMPSV_DUMP_STYLE</strong></td>
<td>Selective dump</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Verbose messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dump off system disk</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compressed</td>
<td></td>
</tr>
<tr>
<td><strong>DMPSB_FLAGS</strong></td>
<td>Additional flags</td>
<td>09</td>
</tr>
<tr>
<td><strong>DMPSV_COMPRESSED</strong></td>
<td>Dump is compressed</td>
<td></td>
</tr>
<tr>
<td><strong>DMPSV_ALPHADUMP</strong></td>
<td>This is an OpenVMS Alpha dump</td>
<td></td>
</tr>
<tr>
<td><strong>DMPSQ_SYSIDENT</strong></td>
<td>System version</td>
<td>&quot;X69G-FT1&quot;</td>
</tr>
<tr>
<td><strong>DMPSQ_LINKTIME</strong></td>
<td>Base image link date/time</td>
<td>&quot;8-JUN-1996 02:07:27.31&quot;</td>
</tr>
<tr>
<td><strong>DMPSL_SYSVER</strong></td>
<td>Base image version</td>
<td>03000000</td>
</tr>
<tr>
<td><strong>DMPSW_DUMPVER</strong></td>
<td>Dump version</td>
<td>0704</td>
</tr>
<tr>
<td><strong>DMPSL_DUMPBLOCKCNT</strong></td>
<td>Count of blocks dumped for memory</td>
<td>0000D3D5</td>
</tr>
<tr>
<td><strong>DMPSL_UNCOMPBLOCKCNT</strong></td>
<td>Uncompressed blocks dumped for memory</td>
<td>0001AEEA</td>
</tr>
<tr>
<td><strong>DMPSL_SAVEPRCCNT</strong></td>
<td>Number of processes saved</td>
<td>00000014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EMBSQ_CR_TIME</strong></td>
<td>Crash date/time</td>
<td>&quot;3-JUL-1996 09:30:13.36&quot;</td>
</tr>
<tr>
<td><strong>EMBSL_CR_CODE</strong></td>
<td>Bugcheck code</td>
<td>&quot;SSRVEXCEPT&quot;</td>
</tr>
<tr>
<td><strong>EMBSB_CR_SCS_NAME</strong></td>
<td>Node name</td>
<td>&quot;SWPCTX&quot;</td>
</tr>
<tr>
<td><strong>EMBST_CR_HW_NAME</strong></td>
<td>Model name</td>
<td>&quot;DEC 3000 Model 400&quot;</td>
</tr>
<tr>
<td><strong>EMBST_CR_LNAME</strong></td>
<td>Process name</td>
<td>&quot;SYSTEM&quot;</td>
</tr>
<tr>
<td><strong>DMPSL_CHECKSUM</strong></td>
<td>Dump header checksum</td>
<td>439E5E91</td>
</tr>
</tbody>
</table>

This example of the SHOW DUMP/HEADER command shows the information in the header.
SHOW EXECUTIVE

Displays the location and size of each loadable image that makes up the executive.

Format

SHOW EXECUTIVE [execlet-name | /SUMMARY]

Parameter

execlet-name
Displays only the data for the specified loadable image. You can use wildcards in execlet-name, in which case SDA displays data for all matching loadable images. The default action is for SDA to display data for all loadable images.

Qualifier

/SUMMARY
Displays a single line of output for all loadable images.

Description

The executive consists of two base images and a number of other executive images.

The base image called SYS$BASE_IMAGE.EXE contains:
- Symbol vectors for universal executive routines and data cells
- Procedure descriptors for universal executive routines
- Globally referenced data cells

The base image called SYS$PUBLIC_VECTORS.EXE contains:
- Symbol vectors for system service procedures
- Procedure descriptors for system services
- Transfer routines for system services

The base images are the pathways to routines and system service procedures in the other executive images.

The SHOW EXECUTIVE command lists the location and size of each executive image. It can enable you to determine whether a given memory address falls within the range occupied by a particular image. (Table 4–1 describes the contents of each executive image.)

SHOW EXECUTIVE also displays the base address and length for each nonzero length image section.

On OpenVMS Alpha the execlets may be sliced. This means each different image section can be relocated in system memory so that they are no longer contiguous. The SHOW EXECUTIVE display contains information on where each image section resides.

The difference between a sliced image and a non-sliced image in the display is that the base, the end, and the length of a sliced image are blank. Only the image section base, end, and length are valid.
There are six different image section types: non-paged read only, non-paged read-write, paged read only, paged read-write, init and fixup. Only the image sections loaded into system memory are displayed.

The MAP command makes it easier to find out in which execlet an address resides. See the description of the MAP command for details.

By default, SDA displays each location within an executive image as an offset from the beginning of the image, for instance, EXCEPTION+00282. Similarly, those symbols that represent system services point to the transfer routine in SYSS$PUBLIC_VECTORS.EXE and not to the actual system service procedure. When tracing the course of a system failure through the listings of modules contained within a given executive image, you may find it useful to load into the SDA symbol table all global symbols and global entry points defined within one or all executive images. See the description of the READ command for additional information.

The SHOW EXECUTIVE command usually shows all components of the executive, as illustrated in the following example. In rare circumstances, you may obtain a partial listing. For instance, once it has loaded the EXCEPTION module (in the INIT phase of system initialization), the system can successfully post a bugcheck exception and save a crash dump before loading all the executive images that are normally loaded.

### Examples

1. SDA> SHOW EXECUTIVE
   VMS Executive layout
   --------------------
<table>
<thead>
<tr>
<th>Image</th>
<th>Base</th>
<th>End</th>
<th>Length</th>
<th>SymVec</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSS$WSDRIVER</td>
<td>A21B200</td>
<td>A21B3600</td>
<td>00001600</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read only</td>
<td>A21B200</td>
<td>A21B3600</td>
<td>00001600</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read/write</td>
<td>A21B6000</td>
<td>A21B6800</td>
<td>00000800</td>
<td></td>
</tr>
<tr>
<td>Linked</td>
<td>5-APR-1998 12:08</td>
<td>LDRIMG 80DA0700</td>
<td>--&lt; not sliced &gt;--</td>
<td></td>
</tr>
<tr>
<td>SYSS$LTDENV</td>
<td>A217A000</td>
<td>A21A8800</td>
<td>0002EB00</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read only</td>
<td>A217A000</td>
<td>A21A8800</td>
<td>0002EB00</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read/write</td>
<td>A21AA000</td>
<td>A21AEB00</td>
<td>0000A00</td>
<td></td>
</tr>
<tr>
<td>Linked</td>
<td>4-APR-1998 22:42</td>
<td>LDRIMG 80D8F600</td>
<td>--&lt; not sliced &gt;--</td>
<td></td>
</tr>
<tr>
<td>LAT$RATING</td>
<td>A2172000</td>
<td>A2172600</td>
<td>0000600</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read only</td>
<td>A2172000</td>
<td>A2172600</td>
<td>0000600</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read/write</td>
<td>A2176000</td>
<td>A2178000</td>
<td>0000A00</td>
<td></td>
</tr>
<tr>
<td>Linked</td>
<td>4-APR-1998 22:45</td>
<td>LDRIMG 80D8F740</td>
<td>--&lt; not sliced &gt;--</td>
<td></td>
</tr>
<tr>
<td>SYSS$OPDRIVER</td>
<td>A216A000</td>
<td>A216D600</td>
<td>00003600</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read only</td>
<td>A216A000</td>
<td>A216D600</td>
<td>00003600</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read/write</td>
<td>A216E000</td>
<td>A216E900</td>
<td>0000A00</td>
<td></td>
</tr>
<tr>
<td>Linked</td>
<td>4-APR-1998 22:56</td>
<td>LDRIMG 80D86C80</td>
<td>--&lt; not sliced &gt;--</td>
<td></td>
</tr>
<tr>
<td>SYSS$OPDRIVER</td>
<td>80022000</td>
<td>80025800</td>
<td>00003800</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read only</td>
<td>9E92F000</td>
<td>9E92F9A00</td>
<td>0000A00</td>
<td></td>
</tr>
<tr>
<td>Nonpaged read/write</td>
<td>9E92F000</td>
<td>9E92F9A00</td>
<td>0000A00</td>
<td></td>
</tr>
<tr>
<td>Linked</td>
<td>4-APR-1998 22:42</td>
<td>LDRIMG 80C1E8C0</td>
<td>--&lt; sliced &gt;--</td>
<td></td>
</tr>
</tbody>
</table>
The SHOW EXECUTIVE command displays the location and length of executive images.

This example displays the use of the wildcard with the SHOW EXECUTIVE command.
3. **SDA> SHOW EXECUTIVE/SUMMARY**

WMS Executive layout summary

<table>
<thead>
<tr>
<th>Image</th>
<th>LDRIMG</th>
<th>Base</th>
<th>End</th>
<th>Length</th>
<th>SymVec</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS$MADDRIVER</td>
<td>80D2A900</td>
<td>83848000</td>
<td>83860000</td>
<td>00018000</td>
<td></td>
</tr>
<tr>
<td>SYS$DADDRIVER</td>
<td>80E00C80</td>
<td>83838000</td>
<td>83848000</td>
<td>00010000</td>
<td></td>
</tr>
<tr>
<td>SYS$LASTDRIVER</td>
<td>80E3C600</td>
<td>8381C000</td>
<td>83838000</td>
<td>0001C000</td>
<td></td>
</tr>
<tr>
<td>SYS$LTDRI VER</td>
<td>80E305C0</td>
<td>837E4000</td>
<td>8381C000</td>
<td>00038000</td>
<td></td>
</tr>
<tr>
<td>LAT$RATING</td>
<td>80E35500</td>
<td>837DC000</td>
<td>837E4000</td>
<td>00008000</td>
<td></td>
</tr>
<tr>
<td>SYS$RTTDRIVER</td>
<td>80DCDF00</td>
<td>837D4000</td>
<td>837DC000</td>
<td>00008000</td>
<td></td>
</tr>
<tr>
<td>SYS$CTDRIVER</td>
<td>80D7BFC0</td>
<td>837C4000</td>
<td>837D4000</td>
<td>00010000</td>
<td></td>
</tr>
<tr>
<td>NDDRIVER</td>
<td>80D86000</td>
<td>8377A000</td>
<td>83782000</td>
<td>00008000</td>
<td></td>
</tr>
<tr>
<td>SYS$FTDRIVER</td>
<td>80DD4280</td>
<td>83772000</td>
<td>8377A000</td>
<td>00008000</td>
<td></td>
</tr>
</tbody>
</table>

This example displays the list of executive images, giving base, end, and length information for those that are not sliced.
SHOW GALAXY

Displays a brief one-page summary of the state of the Galaxy and all the instances in the Galaxy.

Format

SHOW GALAXY

Parameters

None.

Qualifiers

None.

Example

SDA> SHOW GALAXY

Galaxy summary

-----------
GMDB address Creator node ID Revision Creation time State
----------- ------------- ------- --------------------- ---------------
FFFFFFFF.7F234000 00000001 1.0 31-MAR-1999 13:15:08.08 OPERATIONAL
Node ID NODEB address Name Version Join time State
-------- ----------------- -------- -------- --------------------- ---------------
00000000 FFFFFFFF.7F236000 ANDA1A 1.0 31-MAR-1999 14:11:09.08 MEMBER (current instance)
00000001 FFFFFFFF.7F236200 ANDA2A 1.0 31-MAR-1999 14:10:49.06 MEMBER
00000002 FFFFFFFF.7F236400 ANDA3A 1.0 31-MAR-1999 14:13:26.16 MEMBER
00000003 FFFFFFFF.7F236600 - Node block is empty -
SHOW GCT

Displays the contents of the Galaxy configuration tree either in summary (hierarchical) or in detail, node by node.

Format

SHOW GCT [/ADDRESS=n] [/ALL] [/CHILDREN] 
[/HANDLE=n] [/OWNER=n] [/SUMMARY (default)] / [TYPE=n]

Parameters

None.

Qualifiers

/ADDRESS=n
Provides a detailed display of the Galaxy configuration tree (GCT) node at the given address.

/ALL
Provides a detailed display of all nodes in the tree.

/CHILDREN
When used with /ADDRESS=n or /HANDLE=n, the /CHILDREN qualifier causes SDA to display all nodes in the configuration tree that are children of the specified node.

/HANDLE=n
Provides a detailed display of the Galaxy configuration tree (GCT) node with the given handle.

/OWNER=n
Provides a detailed display of all nodes in the tree currently owned by the node with the given handle.

/SUMMARY
Provides a summary display of the Galaxy configuration tree (GCT) in hierarchical form. This qualifier is the default.

/TYPEn=type
Provides a detailed display of all nodes in the tree of the given type, which can be one of the following:

- BUS
- CPU_MODULE
- HARD_PARTITION
- IOP
- PARTITION
- ROOT
- SW_ROOT
- COMMUNITY
- EXP_CHASSIS
- HOSE
- MEMORY_CTRL
- POWER_ENVIR
- SBB
- SYS_CHASSIS
- FRU_DESC
- HW_ROOT
- MEMORY_DESC
- PSEUDO
- SLOT
- SYS_INTER_SWITCH
- FRU_ROOT
- IO_CTRL
- MEMORY_SUB
- RISER
- SMB
- TEMPLATE_ROOT
The type given may be an exact match, in which case just that type is displayed (for example, a CPU); or a partial match, in which case all matching types are displayed (for example, /TYPE=CP displays both CPU and CPU_MODULE nodes).

Examples

1. SDA> SHOW GCT

Galaxy Configuration Tree summary

<table>
<thead>
<tr>
<th>Handle</th>
<th>Hierarchy</th>
<th>Id</th>
<th>Initial Owner</th>
<th>Current Owner</th>
<th>Name/Min PA/Max PA</th>
<th>OS type/Max PA</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>Root</td>
<td>00000000.00000000</td>
<td>00001800</td>
<td>00001800</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>Primary</td>
</tr>
<tr>
<td>00000240</td>
<td>_HW_Root</td>
<td>00000000.00000000</td>
<td>00001800</td>
<td>00001800</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>Primary</td>
</tr>
<tr>
<td>00000280</td>
<td>_IOP</td>
<td>00000000.00000000</td>
<td>00001800</td>
<td>00001800</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>Primary</td>
</tr>
<tr>
<td>00000300</td>
<td>_CPU_Module</td>
<td>00000000.00000000</td>
<td>00001800</td>
<td>00001800</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>Primary</td>
</tr>
<tr>
<td>00000340</td>
<td>_CPU</td>
<td>00000000.00000000</td>
<td>00001800</td>
<td>00001800</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>Primary</td>
</tr>
<tr>
<td>00000380</td>
<td>_CPU_Module</td>
<td>00000000.00000000</td>
<td>00001800</td>
<td>00001800</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>Primary</td>
</tr>
</tbody>
</table>

This command shows the summary (hierarchical) display of the configuration tree.
2. **SDA> SHOW GCT/HANDLE=00000700**

**Galaxy Configuration Tree**

Handle: 0000700  Address: FFFFFFFF.83694740
Node type: Memory_Sub  Size: 0080
Id: 00000000.00000000  Flags: 00000000.00000001  Hardware

**Related nodes:**

<table>
<thead>
<tr>
<th>Node relationship</th>
<th>Handle</th>
<th>Type</th>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial owner</td>
<td>00001580</td>
<td>Community</td>
<td>00000000.00000000</td>
</tr>
<tr>
<td>Parent</td>
<td>&lt;Same&gt;</td>
<td>HW_Root</td>
<td>00000000.00000000</td>
</tr>
<tr>
<td>Previous sibling</td>
<td>00000640</td>
<td>CPU_Module</td>
<td>00000000.00000003</td>
</tr>
<tr>
<td>Next sibling</td>
<td>&lt;None&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child</td>
<td>00000780</td>
<td>Memory_Ctrl</td>
<td>00000000.00000005</td>
</tr>
<tr>
<td>Configuration binding</td>
<td>00000240</td>
<td>HW_Root</td>
<td>00000000.00000000</td>
</tr>
<tr>
<td>Affinity binding</td>
<td>00000240</td>
<td>HW_Root</td>
<td>00000000.00000000</td>
</tr>
</tbody>
</table>

Min. physical address: 00000000.0000000
Max. physical address: 00000000.FFFFFFFF

This command shows the detailed display of the specified node.
**SHOW GLOBAL_SECTION_TABLE, SHOW GST**

Displays information contained in the global section table.

**Format**

SHOW GLOBAL_SECTION_TABLE or SHOW GST [/SECTION_INDEX=n]

**Parameters**

None.

**Qualifiers**

/SECTION_INDEX=n
Displays only the global section table entry for the specified section.

**Description**

Displays the entire contents of the global section table, unless you specify the qualifier /SECTION_INDEX. This command is equivalent to SHOW PROCESS/PROCESS_SECTION_TABLE/SYSTEM. See the SHOW PROCESS command and Table 4–18 for more information.
**Example**

SDA> SHOW GST

Global Section Table

Global section table information

Last entry allocated          00000010
First free entry             00000000

Global section table

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
<th>Sect/GTE Addr</th>
<th>Pagelets</th>
<th>Window</th>
<th>VBN</th>
<th>CCB/GSD</th>
<th>Refcnt</th>
<th>Flink</th>
<th>Blink</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>81409F08</td>
<td>FFFFFFFF.835E8000</td>
<td>00000025</td>
<td>81413960</td>
<td>000000003</td>
<td>000000000</td>
<td>00000003</td>
<td>0000 0000</td>
<td>AMOD=KRNL</td>
<td></td>
</tr>
<tr>
<td>00000002</td>
<td>81409F80</td>
<td>FFFFFFFF.83A10000</td>
<td>00000063</td>
<td>81419B40</td>
<td>000000208</td>
<td>000000000</td>
<td>00000007</td>
<td>0000 0000</td>
<td>AMOD=KRNL</td>
<td></td>
</tr>
<tr>
<td>00000003</td>
<td>81409F88</td>
<td>FFFFFFFF.83A10000</td>
<td>0000000C</td>
<td>8142ED00</td>
<td>000000004</td>
<td>826B2DD0</td>
<td>000000000</td>
<td>0003 0003</td>
<td>WRTMOD=EXEC AMOD=USER PERM SYSGRL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B2D50_002</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]SYS$SISHR.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000004</td>
<td>81409F60</td>
<td>FFFFFFFF.83A10000</td>
<td>00000011</td>
<td>81432F80</td>
<td>000000022</td>
<td>000000000</td>
<td>00000002</td>
<td>0000 0000</td>
<td>AMOD=KRNL</td>
<td></td>
</tr>
<tr>
<td>00000005</td>
<td>81409F90</td>
<td>FFFFFFFF.83A10000</td>
<td>00000026</td>
<td>8142D080</td>
<td>000000018</td>
<td>826B3320</td>
<td>000000000</td>
<td>0005 0005</td>
<td>WRTMOD=EXEC AMOD=USER PERM SYSGRL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B32A0_006</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DISMNTSHR.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000006</td>
<td>81409F10</td>
<td>FFFFFFFF.83A50000</td>
<td>00000084</td>
<td>8144E480</td>
<td>000000003</td>
<td>000000000</td>
<td>00000000</td>
<td>0000 0000</td>
<td>AMOD=KRNL</td>
<td></td>
</tr>
<tr>
<td>00000007</td>
<td>81409E40</td>
<td>FFFFFFFF.83A76000</td>
<td>00000038</td>
<td>8144E480</td>
<td>000000000</td>
<td>000000000</td>
<td>00000001</td>
<td>0000 0000</td>
<td>CRF WRT AMOD=KRNL</td>
<td></td>
</tr>
<tr>
<td>00000008</td>
<td>81409E40</td>
<td>FFFFFFFF.83A76000</td>
<td>0000007A</td>
<td>8144F280</td>
<td>000000004</td>
<td>826B37A0</td>
<td>000000000</td>
<td>0008 0008</td>
<td>WRTMOD=EXEC AMOD=USER PERM SYSGRL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B36B0_001</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DTI$SHARE.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000009</td>
<td>81409E90</td>
<td>FFFFFFFF.83A04000</td>
<td>000000A0</td>
<td>81402F80</td>
<td>000000091</td>
<td>826B37F0</td>
<td>000000001</td>
<td>0009 0009</td>
<td>WRTMOD=EXEC AMOD=USER PERM SYSGRL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B36B0_005</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DTI$SHARE.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000A</td>
<td>81409EF0</td>
<td>FFFFFFFF.83A04000</td>
<td>00000010</td>
<td>00000000</td>
<td>000000000</td>
<td>826B40F0</td>
<td>000000000</td>
<td>000A 000A</td>
<td>DZRO WRTMOD=EXEC AMOD=USER PERM SYSGRL PAGFIL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B36B0_001</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DTI$SHARE.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000B</td>
<td>81409EF0</td>
<td>FFFFFFFF.83A04000</td>
<td>00000010</td>
<td>00000000</td>
<td>000000000</td>
<td>826B40F0</td>
<td>000000000</td>
<td>000A 000A</td>
<td>DZRO WRTMOD=EXEC AMOD=USER PERM SYSGRL PAGFIL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B36B0_005</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DTI$SHARE.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000C</td>
<td>81409EF0</td>
<td>FFFFFFFF.83A04000</td>
<td>00000010</td>
<td>00000000</td>
<td>000000000</td>
<td>826B40F0</td>
<td>000000000</td>
<td>000A 000A</td>
<td>DZRO WRTMOD=EXEC AMOD=USER PERM SYSGRL PAGFIL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B36B0_001</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DTI$SHARE.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000D</td>
<td>81409EF0</td>
<td>FFFFFFFF.83A04000</td>
<td>00000010</td>
<td>00000000</td>
<td>000000000</td>
<td>826B40F0</td>
<td>000000000</td>
<td>000A 000A</td>
<td>DZRO WRTMOD=EXEC AMOD=USER PERM SYSGRL PAGFIL</td>
<td></td>
</tr>
<tr>
<td>Name = INS$826B36B0_005</td>
<td>File = DISK$X901_K5L:[VMS$COMMON.SYSLIB]DTI$SHARE.EXE;1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SHOW GLOCK

Displays the Galaxy locks for the Galaxy Management Database (GMDB), process tables, and/or system tables.

Format

SHOW GLOCK [/BRIEF] 
[/GMDB_TABLE] 
[/PROCESS_TABLE [=n]] 
[/SYSTEM_TABLE [=n]] 
[/ALL] 
[/ADDRESS=n [/PHYSICAL]] 
[/HANDLE=n [/LINKED]]

Parameters

None.

Qualifiers

/BRIEF
Displays a single line for each Galaxy lock, regardless of any other qualifiers.

/GMDB_TABLE
Displays the Galaxy lock table for the Galaxy Management Database (GMDB) including the embedded and attached Galaxy locks.

/PROCESS_TABLE [=n]
Displays all the process Galaxy lock tables with the embedded and attached Galaxy locks, as well as a summary table. The /PROCESS_TABLE=n qualifier displays the single Galaxy lock table without a summary page.

/SYSTEM_TABLE [=n]
Displays all the system Galaxy lock tables with the embedded and attached Galaxy locks, as well as a summary table. The /SYSTEM_TABLE=n qualifier displays the single Galaxy lock table without a summary page.

/ALL
Displays information provided by the /GMDB_TABLE, /PROCESS_TABLE, and /SYSTEM_TABLE qualifiers. The /ALL qualifier also displays information from the base GMDB Galaxy lock.

/ADDRESS=n [/PHYSICAL]
Displays the single Galaxy lock at address n. Because process Galaxy locks are located by their physical address, you must use the /PHYSICAL qualifier to enter such an address.

/HANDLE=n [/LINKED]
Displays the single Galaxy lock whose handle is n. The optional qualifier /LINKED causes SDA to display all Galaxy locks linked to the one specified.
Examples

1. SDA> SHOW GLOCK
   Galaxy Lock Database
   ------------------------
   Base address of GLock segment of GMDB: FFFFFFFF.7F238000
   Length: 00000000.00082000
   Nodes: 00000000.00000007 Flags: 00000000.00000000
   Process tables: 00000000.00000400 System tables: 00000000.00000400
   First free: 00000002 First used: 00000001
   First used: 00000001
   Embedded GLocks:
   GLock address: FFFFFFFF.7F238020 Handle: 80000000.00000805
   GLock name: GMDB_GLOCK_LOCK Flags: 00
   Owner count: 00 Owner node: 00
   Node sequence: 0000 Owner: 000000
   IPL: 08 Previous IPL: 00
   Wait bitmask: 00000000.00000000 Timeout: 00000000
   Thread ID: 00000000.00000000
   GLock address: FFFFFFFF.7F23D10 Handle: 80000000.0000083B
   GLock name: SYS_LCKTBL_LOCK Flags: 00
   Owner count: 00 Owner node: 00
   Node sequence: 0000 Owner: 000000
   IPL: 08 Previous IPL: 00
   Wait bitmask: 00000000.00000000 Timeout: 00000000
   Thread ID: 00000000.00000000
   This example shows the summary of the Galaxy lock database.

2. SDA> SHOW GLOCK/PROCESS_TABLE
   Galaxy Lock Database: Process Lock Table #0001
   ----------------------------
   Base address of Process Lock Table #0001: FFFFFFFF.7F23A000
   Lock size: 0040 Flags: 01 VALID
   Region Index/Sequence: 0008/00000001 Access mode: 03
   Region physical size: 00000000.00000200 Virtual size: 00000000.00000200
   Number of locks: 00000000.00000080 Nodes: 00000000.00000007
   Per-node reference counts:
     Node  Count
     ----  ----
     0000  0001
     0001  0001
     0002  0001
   Embedded GLock:
   GLock address: FFFFFFFF.7F23A040 Handle: 80000000.00000C09
SDA Commands
SHOW GLOCK

GLock name: PLCKTBL_LOCK001 Flags: 00
Owner count: 00 Owner node: 00
Node sequence: 0000 Owner: 000000
IPL: 00 Previous IPL: 00
Wait bitmask: 00000000.00000000 Timeout: 00000000
Thread ID: 00000000.00000000

Attached GLocks:

GLock address: P00000000.C05EC7C0 Handle: 00000001.000000F9
GLock name: CPU_BAL_LOCK Flags: 00
Owner count: 00 Owner node: 00
Node sequence: 0000 Owner: 000000
IPL: 00 Previous IPL: 00
Wait bitmask: 00000000.00000000 Timeout: 00000000
Thread ID: 00000000.00000000

GLock address: P00000000.C05EC000 Handle: 00000001.00000001
GLock name: CPU_BAL_LOCK Flags: 00
Owner count: 00 Owner node: 00
Node sequence: 0000 Owner: 000000
IPL: 00 Previous IPL: 00
Wait bitmask: 00000000.00000000 Timeout: 00000000
Thread ID: 00000000.00000000

Used GLock count = 0020
Free GLock count = 0060

Galaxy Lock Database: Process Lock Table Summary
------------------------------------------------------------------------
Total used Process Lock Tables: 00000001
Total free Process Lock Tables: 000003FF

This example shows the Galaxy locks for all processes.
SHOW GMDB

Displays the contents of the Galaxy management data base (GMDB) and/or the node blocks of the instances in the Galaxy system.

Format

SHOW GMDB [/ALL]

[/NODE [=name | =n] /ADDRESS=n] [/SUMMARY]

Parameters

None.

Qualifiers

/ADDRESS

Specifies the address of a single node block to be displayed when used with the /NODE qualifier. See the description of the /NODE qualifier.

/ALL

Displays the contents of the Galaxy Management Database and all node blocks that have ever been used (contents nonzero).

/NODE [=name | =n] /ADDRESS=n

Displays the contents of the specified node block, given by either the name of the instance, the partition number, or the address of the node block. If the /NODE qualifier is given alone, then the node block for the current instance is displayed.

/SUMMARY

Displays a one-page summary of the GMDB and all node blocks.

Note

The default action displays the contents of the Galaxy Management Database.

Examples

1. SDA> SHOW GMDB

Galaxy Management Database

Base address of GMDB: FFFFFFFF.7F234000
Base address of NODEB for this instance: FFFFFFFF.7F236000

Revision: 1.0 Maximum node ID: 00000003
Creation time: 31-MAR-1999 13:15:08.08 Incarnation: 00000000.00000003
State: OPERATIONAL Creator node: 00000001
Base size: 00000000.00004000 Total size: 00000000.000A6000
Last joiner ID: 00000002 Remover node ID: FFFFFFFF
Last leaver ID: 00000002 Node timeout (msec) 5000.
Lock owner: 00000002 Lock flags: 0000
Break owner: FFFFFFFF Breaker ID: FFFFFFFF

Version Information:

Min Version Operational 1.0 Min Version Allowed 1.0
Max Version Operational 1.0

Membership bitmask: FFFFFFFF.7F236800
This example shows the overall summary of the galaxy management database.

2. SDA> SHOW GMDB/NODE=0

GMDB: Node ID 00000000 (current instance)

-- Base address of node block: FFFFFFFF.7F236000

    Version: 1.0 Node name: ANDA1A
    Join time: 31-MAR-1999 14:11:09.08 Incarnation: 00000000.00000005
    State: MEMBER Crash_all acknowledge: 00000000
    Validation done: 00000000 Reform done: 00000000
    IP interrupt mask: 00000000.00000000
    Little brother: 00000002 Heartbeat: 00000000.0019EAD1
    Big brother: 00000001 Last watched_node: 00000000
    Watched_node #0: FFFFFFFF.7F236078 Node watched: 00000002
    Last heartbeat: 00000000.0017C1AD Miss count: 00000000)

This example shows galaxy management database information for the specified instance.
SHOW GSD

Displays information contained in the global section descriptors.

Format

SHOW GSD [/ADDRESS=n] /ALL /DELETED /GLXGRP
/GLXSYS /GROUP /SYSTEM

Parameters

None.

Qualifiers

/ADDRESS=n
Displays a specific global section descriptor entry, given its address.

/ALL
Displays information in all the global section descriptors, that is, the system,
group, and deleted global section descriptors, plus the Galaxy group and Galaxy
system global section descriptors, if the system or dump being analyzed is a
member of an OpenVMS Galaxy system. This qualifier is the default.

/DELETED
Displays information in the deleted (that is, delete pending) global section
descriptors.

/GLXGRP
Displays information in the group global section descriptors of a Galaxy system.

/GLXSYS
Displays information in the system global section descriptors of a Galaxy system.

/GROUP
Displays information in the group global section descriptors.

/SYSTEM
Displays information in the system global section descriptors.

Description

The SHOW GSD command displays information that resides in the global section
descriptors. Table 4-3 shows the fields and their meaning.
Table 4-3  GSD Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>Gives the address of the global section descriptor</td>
</tr>
<tr>
<td>NAME</td>
<td>Gives the name of the global section</td>
</tr>
<tr>
<td>GSTX</td>
<td>Gives the global section table index</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Gives the settings of flags for specified global section, as a hexadecimal number; also displays key flag bits by name</td>
</tr>
<tr>
<td>BASEPFN1</td>
<td>Gives physical page frame number at which the section starts</td>
</tr>
<tr>
<td>PAGES1</td>
<td>Gives number of pages (not pagelets) in section</td>
</tr>
<tr>
<td>REFCNT1</td>
<td>Gives number of times this global section is mapped</td>
</tr>
</tbody>
</table>

1This field only applies to PFN mapped global sections.

Example

SDA > SHOW GSD

System Global Section Descriptor List
----------------------------------------- ----------- PFNMAP----------
ADDRESS NAME GSTX FLAGS REFCNT PAGES BASEPFN
----------------------------------------- -----------
817DAF30 SECDIX_422 02DD 0082C3C9 WRT AMOD=USER FERM
817DAE60 SECDIX_421 02DC 008A83CD DZRO WRT AMOD=USER PAGFIL
817DAD90 SECDIX_420 02DB 0088C3CD DZRO WRT AMOD=USER PAGFIL PAGFIL
817DACC0 SECDIX_419 02DA 008883DC DZRO WRT AMOD=USER PAGFIL PAGFIL
817DABE0 SECDIX_418 0000 0001C3C1 AMOD=USER PAGFIL 0000080B 00000002 00000000
817DAB60 SECDIX_417 0000 0001C3C1 AMOD=USER PAGFIL 0000080B 00000002 00000000
817DA890 SECDIX_412 02D6 0080C3CD DZRO WRT AMOD=USER PAGFIL PAGFIL
817DA850 SECDIX_411 02D5 008083CD DZRO WRT AMOD=USER PAGFIL PAGFIL

ZK-8830A-GE
SHOW HEADER

Displays the header of the dump file.

Format
SHOW HEADER

Parameters
None.

Qualifiers
None.

Description
The SHOW HEADER command produces a 10-column display, each line of which displays both the hexadecimal and ASCII representation of the contents of the dump file header in 32-byte intervals. Thus, the first eight columns, when read right to left, represent the hexadecimal contents of 32 bytes of the header; the ninth column, when read left to right, records the ASCII equivalent of the contents. (Note that the period [.] in this column indicates an ASCII character that cannot be displayed.)

After it displays the contents of the header blocks, the SHOW HEADER command displays the hexadecimal contents of the saved error log buffers.

See the OpenVMS AXP Internals and Data Structures manual for a discussion of the information contained in the dump file header. See also the SHOW DUMP and CLUE ERRLLOG commands, which can be used to obtain formatted displays of the dump header and error log buffers.

Example
SDA> SHOW HEADER

Dump file header

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hexadecimal</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>7FFA6000 00000000 7FFA1C98 00000000 0000187C 08000FC1 00000004</td>
<td>....Á...</td>
</tr>
<tr>
<td>00001FFF</td>
<td>00000000 00002000 8000A000 00000000 7AFFBAD0 00000000 7FFAC100</td>
<td>.Â....,*z...B..</td>
</tr>
<tr>
<td>00008162</td>
<td>00000000 00000001 00000000 00040704 FF000000 00000000 7C13670</td>
<td>p6À....*..</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td>...........í...</td>
</tr>
<tr>
<td>FF7PC000</td>
<td>FF000000 50000000 00000000 00000000 00000000</td>
<td>...........í...</td>
</tr>
</tbody>
</table>

Saved error log messages

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hexadecimal</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000 00000000 00000001 00000000 00000000 00000000 00000000</td>
<td>.b...dp...   b...</td>
</tr>
<tr>
<td>001ACB02</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td>.c...SWPCTX  &quot;.&quot;</td>
</tr>
<tr>
<td>00303320</td>
<td>34354412 00000002 00000200 00000000 00000000 00000000 00000000</td>
<td>.11...X691-FT1...DEC 300</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td>.F1,c...X691-FT1</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td>.00,0,y,l...X691-FT1</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td>.DEC 3000 Model 400...</td>
</tr>
<tr>
<td>00000000</td>
<td>00000000 00000000 00000000 00000000 00000000 00000000 00000000</td>
<td>.w...swpctx6DK</td>
</tr>
</tbody>
</table>

The SHOW HEADER command displays the contents of the dump file's header. Ellipses indicate hexadecimal information omitted from the display.
SHOW LAN

Displays information contained in various local area network (LAN) data structures.

Format

SHOW LAN [/qualifier[,...]]

Parameters

None.

Qualifiers

/CLIENT=name
Specifies that information be displayed for the specified client. Valid client designators are SCA, DECNET, LAT, MOPRC, TCPIP, DIAG, ELN, BIOS, LAST, USER, ARP, MOPDL, LOOP, BRIDGE, DNAME, ENCRY, DTIME, and LTM. The /CLIENT, /DEVICE, and /UNIT qualifiers are synonymous and mutually exclusive.

/CLUEXIT
Specifies that cluster protocol information be displayed.

/COUNTERS
Specifies that the LAN station block (LSB) and unit control block (UCB) counters be displayed.

/CSMACD
Specifies that Carrier Sense Multiple Access with Collision Detect (CSMA/CD) information for the LAN be displayed. By default, both CSMA/CD and Fiber Distributed Data Interface (FDDI) information is displayed.

/DEVICE=name
Specifies that information be displayed for the specified device, unit, or client. For each LAN adapter on the system there is one device called units or clients. Device designators are specified in the format XXdn, where XX is the type of device, d is the device letter, and n is the unit number. The device letter and unit number are optional. The first unit, which is always present, is the template unit. These are specified as indicated in this example, for a DEMNA which is called EX:

/DEVICE=EX—display all EX devices on the system
/DEVICE=EXA—display the first EX device only
/DEVICE=EXA0—display the first EXA unit
/DEVICE=SCA—display SCA unit
/DEVICE=LAT—display LAT units

Valid client names are listed in the /CLIENT=name qualifier. The /CLIENT, /DEVICE, and /UNIT qualifiers are synonymous and mutually exclusive.

/ELAN
Specifies information from an Emulated LAN (ELAN) that runs over an asynchronous transfer mode (ATM) network. The /ELAN qualifier displays the LAN Station Block (LSB) address, device state, and the LSB fields pertinent
SHOW LAN

to an ELAN for both the parent ATM device and the ELAN pseudo-device drivers. It also specifies the name, description, parent device, state, and LAN emulation client (LEC) attributes of the ELAN.

The qualifier /ELAN used with the device qualifier (/LAN/device=ELA) will only display information for the specified device or pseudo-device.

/ERRORS
Specifies that the LSB and UCB error counters be displayed.

/FDDI
Specifies that Fiber Distributed Data Interface (FDDI) information for the LAN be displayed. By default, both CSMA/CD and FDDI information is displayed.

/FULL
Specifies that all information from the LAN, LSB, and UCB data structures be displayed.

/ICOUNTERS
Specifies internal counters of the drivers by displaying the internal counters. If the /ICOUNTERS qualifier is used with the /DEVICE qualifier, the /ICOUNTERS specifies the internal counters of a specific driver.

/QUEUE
Specifies a listing of all queues, whether their status is valid or invalid, and all elements of the queues. If the /QUEUE qualifier is used with the /DEVICE qualifier, the /QUEUE specifies a specific queue.

/SUMMARY
Specifies that only a summary of LAN information (a list of flags, LSBs, UCBs, and base addresses) be printed. This is the default.

/TIMESTAMPS
Specifies that time information (such as start and stop times and error times) from the device and unit data structures be printed. SDA displays the data in chronological order.

/UNIT=name
Specifies that information be displayed for the specified unit. See the descriptions for /CLIENT=name and /DEVICE=name qualifiers.

/VCI
Specifies that information be displayed for the VMS Communication Interface Block (VCIB) for each LAN device with an active VCI user. If you use the /VCI qualifier with the /DEVICE qualifier, the VCIB is only displayed for the specified device.

Description

The SHOW LAN command displays information contained in various local area network (LAN) data structures. By default, or when the /SUMMARY qualifier is specified, SHOW LAN displays a list of flags, LSBs, UCBs, and base addresses. When the /FULL qualifier is specified, SHOW LAN displays all information found in the LAN, LSB, and UCB data structures.
Examples

1. SDA> SHOW LAN/FULL

LAN Data Structures
-------------------

-- LAN Information Summary 23-MAY-1996 13:07:52 --

LAN flags: 00000004 LAN_INIT

LAN block address 80DB7140 Timer DELTA time 10000000
Number of stations 2 DAT sequence number 1
LAN module version 1 First SVAPEE FFDF60F0
LANIDEF version 51 Number of PTEs 3
LANUDEF version 26 SVA of first page 8183C000
First LSB address 80DCA980


Creation time None Times created 0
Deletion time None Times deleted 0
Module EAB 00000000 Latest EIB 00000000
Port EAB 00000000
Station EAB 00000000
NM flags: 00000000


Creation time None Times created 0
Deletion time None Times deleted 0
Module EAB 00000000 Link EAB 00000000
Port EAB 00000000 PHY port EAB 00000000
Station EAB 00000000 Module EIB 00000000
NM flags: 00000000

LAN Data Structures
-------------------


LSB address 80DCA980 Driver code address 80CAE838
Driver version 00000001.07010037 Device1 code address 00000000
Devicel version 00000000.00000000 Device2 code address 00000000
Device2 version 00000000.00000000 LAN code address 80CAFA00
LAN version 00000000.07010112 DLL type CSMACD
Device name EY_NITC2 MOP name MXE
MOP ID 94 HW serial Not supplied
HW version 00000000 Promiscuous mode OFF
Controller mode NORMAL Promiscuous UCB 00000000
Internal loopback OFF All multicast state OFF
Hardware address 08-00-03-DE-00-12 CRC generation mode ON
Physical address AA-00-04-00-88-FE Full Duplex Enable OFF
Active unit count 1 Full Duplex State OFF
Line speed 10

Flags: 00000000
Char: 00000000
Status: 00000003 RUN,INITED
LAN Data Structures
--------------------------
-- ESA Device Information (cont) 23-MAY-1996 13:07:52 --

Put rcv ptr/index 00000000
Put xmt ptr/index 80DCB620
Put cmd ptr/index 00000000
Put uns ptr/index 00000000
Put smt ptr/index 00000000

RBufs owned by dev 0
XEnts owned by dev 0
CEnts owned by dev 0
UEnts owned by dev 0
SEnts owned by dev 0

Current rcv buffers 17
Rgst MAX rcv buffers 32
Rgst MIN rcv buffers 16
Curr MAX rcv buffers 32
Curr MIN rcv buffers 16
FILL rcv buffers 16
ADD rcv buffers 32

LAN Data Structures
--------------------------
-- ESA Device Information (cont) 23-MAY-1996 13:07:52 --

Last receive 23-MAY 13:07:51
Last transmit 23-MAY 13:07:50
ADP address 80D4B280
DAT stage 00000000
DAT number started 1
DAT number failed 0
DAT VCRP 80DCBB80
Mailbox enable flag 0
CSR base phys addr 00000000

LAN Data Structures
--------------------------

Creation time None
Deletion time None
Enabled time None
Disabled time None
EIB address 00000000
LLB address 00000000
LHB address 00000000
First LPB address 00000000

LAN Data Structures
--------------------------

ISR FKB sched 23-MAY 13:07:51
ISR FKB time 23-MAY 13:07:51
IPL8 FKB sched 23-MAY 13:07:20
IPL8 FKB time 23-MAY 13:07:20
RESET FKB sched None
RESET FKB time None
NM FKB sched None
NM FKB time None

Fork status code 0
LAN Data Structures

### ESA Queue Information 23-MAY-1996 13:07:52 --

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Queue ID</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control hold queue</td>
<td>80DCACF0</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Control request queue</td>
<td>80DCACF8</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Control pending queue</td>
<td>80DCAD00</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Transmit request queue</td>
<td>80DCACE8</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Transmit pending queue</td>
<td>80DCAD18</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Receive buffer list</td>
<td>80DCAD38</td>
<td>Valid, 17 elements</td>
</tr>
<tr>
<td>Receive pending queue</td>
<td>80DCAD20</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Post process queue</td>
<td>80DCAD08</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Delay queue</td>
<td>80DCAD10</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Auto restart queue</td>
<td>80DCAD28</td>
<td>Valid, empty</td>
</tr>
<tr>
<td>Netwrk mgmt hold queue</td>
<td>80DCAD30</td>
<td>Valid, empty</td>
</tr>
</tbody>
</table>

### ESA Multicast Address Information 23-MAY-1996 13:07:52 --

- AB-00-00-04-00-00

### ESA Unit Summary 23-MAY-1996 13:07:52 --

<table>
<thead>
<tr>
<th>UCB</th>
<th>UCB Addr</th>
<th>Fmt</th>
<th>Value</th>
<th>Client State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESA0</td>
<td>80D4F6C0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESA1</td>
<td>80E35400</td>
<td>Eth</td>
<td>60-03</td>
<td>DECNET 0017 STRTN,LEN,UNIQ,STRID</td>
</tr>
</tbody>
</table>

LAN Data Structures

### ESA Counters Information 23-MAY-1996 13:07:52 --

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octets received</td>
<td>596</td>
</tr>
<tr>
<td>PDUs received</td>
<td>8</td>
</tr>
<tr>
<td>Mcast octets received</td>
<td>596</td>
</tr>
<tr>
<td>Mcast PDUs received</td>
<td>8</td>
</tr>
<tr>
<td>Unrec indiv dest PDUs</td>
<td>0</td>
</tr>
<tr>
<td>Unrec mcast dest PDUs</td>
<td>1</td>
</tr>
<tr>
<td>Data overruns</td>
<td>0</td>
</tr>
<tr>
<td>Unavail station buffers</td>
<td>0</td>
</tr>
<tr>
<td>Unavail user buffers</td>
<td>0</td>
</tr>
<tr>
<td>CRC errors</td>
<td>0</td>
</tr>
<tr>
<td>Alignment errors</td>
<td>0</td>
</tr>
<tr>
<td>Rcv data length err</td>
<td>0</td>
</tr>
<tr>
<td>Frame size errors</td>
<td>0</td>
</tr>
<tr>
<td>Frames too long</td>
<td>0</td>
</tr>
<tr>
<td>Seconds since zeroed</td>
<td>34</td>
</tr>
<tr>
<td>Station failures</td>
<td>0</td>
</tr>
</tbody>
</table>

Octets sent: 230  
Mcast octets sent: 138  
Mcast PDUs sent: 3  
PDUs sent: 5  
Unrec indiv dest PDUs sent: 0  
Unrec mcast dest PDUs sent: 0  
Data overruns sent: 0  
Unavail station buffers sent: 0  
Unavail user buffers sent: 0  
CRC errors sent: 0  
Alignment errors sent: 0  
Rcv data length err sent: 0  
Frame size errors sent: 0  
Frames too long sent: 0  
Seconds since zeroed sent: 34  
Station failures sent: 0
LAN Data Structures
-------------------
-- ESA Counters Information (cont) 23-MAY-1996 13:07:52 --

No work transmits 0
Buffer_Addr transmits 0
SVAPTE/BOFF transmits 0
Global page transmits 0
Bad PTE transmits 0
Restart pending counter 0
+00 MCA not enabled 187
+04 Xmt underflows 0
+08 Rcv overflows 0
+0C Memory errors 0
+10 Babbling errors 0
+14 Local buffer errors 0
+18 LANCE interrupts 202
+1C Xmt ring <31:0> 00000000
+20 Xmt ring <63:32> 00000000
+24 Soft errors handled 0
+28 Generic (or unused) 802CAD18
+30 Generic (or unused) 00000000
+34 Generic (or unused) 00000000
+38 Generic (or unused) 00000000
+40 Generic (or unused) 00000000
+44 Generic (or unused) 61616161
+48 Generic (or unused) 61616161
+50 Generic (or unused) 61616161
+54 Generic (or unused) 61616161

LAN Data Structures
-------------------

Fatal error count 0
Fatal error code None
Prev error code None
Transmit timeouts 0
Control timeouts 0
Restart failures 0
Power failures 0
Bad PTE transmits 0
Loopback failures 0
System ID failures 0
ReqCounters failures 0

LAN Data Structures
-------------------
-- ESA0 Template Unit Information 23-MAY-1996 13:07:52 --

LSB address 80DCA980
VCIB address 00000000
Stop IRP address 00000000
Restart IRP address 00000000
LAN medium CSMACD
Packet format Ethernet
Eth protocol type 00-00
802E protocol ID 00-00-00-00-00
802.2 SAP 00
802.2 Group SAPs 00,00,00,00
Controller mode NORMAL
Internal loopback OFF
CRC generation mode ON
Functional Addr mod ON
Hardware address 08-00-03-DE-00-12
Physical address FF-FF-FF-FF-FF

Fatal error count 0
Last error CSR 00000000
Last fatal error None
Prev fatal error None
Last USB time None
Last UUB time None
Last CRC time None
Last CRC srcadr None
Last length erro None
Last exc collisi None
Last carrier fai None
Last late collis None

Error count 0
Parameter mask 00000000
Promiscuous mode OFF
All multicast mode OFF
Source Routing mode TRANSPARENT
Access mode EXCLUSIVE
Shared user DES None
Padding mode OFF
Automatic restart DISABLED
802.2 service 802.2 service User
Rcv buffers to save 1
Minimum rcv buffers 4
User transmit FC/AC ON
User receive FC/AC OFF
LAN Data Structures
-------------------
-- ESA1 60-03 (DECNET) Unit Information 23-MAY-1996 13:07:52 --

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSB address</td>
<td>80DCA980</td>
</tr>
<tr>
<td>Error count</td>
<td>0</td>
</tr>
<tr>
<td>VCIB address</td>
<td>00000000</td>
</tr>
<tr>
<td>Parameter mask</td>
<td>00DA8695</td>
</tr>
<tr>
<td>Stop IRP address</td>
<td>80E047C0</td>
</tr>
<tr>
<td>Promiscuous mode</td>
<td>OFF</td>
</tr>
<tr>
<td>Restart IRP address</td>
<td>00000000</td>
</tr>
<tr>
<td>All multicast mode</td>
<td>OFF</td>
</tr>
<tr>
<td>LAN medium</td>
<td>CSMACD</td>
</tr>
<tr>
<td>Source Routing mode</td>
<td>TRANSPARENT</td>
</tr>
<tr>
<td>Packet format</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Access mode</td>
<td>EXCLUSIVE</td>
</tr>
<tr>
<td>Eth protocol type</td>
<td>60-03</td>
</tr>
<tr>
<td>Shared user DES</td>
<td>None</td>
</tr>
<tr>
<td>802.2 protocol ID</td>
<td>00-00-00-00-00</td>
</tr>
<tr>
<td>Padding mode</td>
<td>ON</td>
</tr>
<tr>
<td>802.2 SAP</td>
<td>00</td>
</tr>
<tr>
<td>Automatic restart</td>
<td>DISABLED</td>
</tr>
<tr>
<td>802.2 Group SAPs</td>
<td>00,00,00,00</td>
</tr>
<tr>
<td>Allow prom client</td>
<td>ON</td>
</tr>
<tr>
<td>Controller mode</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Can change address</td>
<td>OFF</td>
</tr>
<tr>
<td>Internal loopback</td>
<td>OFF</td>
</tr>
<tr>
<td>802.2 service</td>
<td>User</td>
</tr>
<tr>
<td>CRC generation mode</td>
<td>ON</td>
</tr>
<tr>
<td>Rcv buffers to save</td>
<td>10</td>
</tr>
<tr>
<td>Functional Addr mod</td>
<td>ON</td>
</tr>
<tr>
<td>Minimum rcv buffers</td>
<td>4</td>
</tr>
<tr>
<td>Hardware address</td>
<td>08-00-03-DE-00-12</td>
</tr>
<tr>
<td>User transmit FC/AC</td>
<td>ON</td>
</tr>
<tr>
<td>Physical address</td>
<td>AA-00-04-00-88-FE</td>
</tr>
<tr>
<td>User receive FC/AC</td>
<td>OFF</td>
</tr>
</tbody>
</table>

LAN Data Structures
-------------------
-- ESA1 60-03 (DECNET) Counters Information 23-MAY-1996 13:07:52 --

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octets received</td>
<td>483</td>
</tr>
<tr>
<td>Octets sent</td>
<td>180</td>
</tr>
<tr>
<td>PDUs received</td>
<td>7</td>
</tr>
<tr>
<td>PDUs sent</td>
<td>3</td>
</tr>
<tr>
<td>Mcast octets received</td>
<td>483</td>
</tr>
<tr>
<td>Mcast octets sent</td>
<td>180</td>
</tr>
<tr>
<td>Mcast PDUs received</td>
<td>7</td>
</tr>
<tr>
<td>Mcast PDUs sent</td>
<td>3</td>
</tr>
<tr>
<td>Unavail user buffer</td>
<td>0</td>
</tr>
<tr>
<td>Multicast not enabled</td>
<td>0</td>
</tr>
<tr>
<td>Last UUB time</td>
<td>None</td>
</tr>
<tr>
<td>User buffer too small</td>
<td>0</td>
</tr>
</tbody>
</table>

The SHOW LAN/FULL command displays information for all LAN, LSB, and UCB data structures.
2. **SDA> SHOW LAN/TIME**

   -- LAN History Information 12-FEB-1995 11:08:48 --

   12-FEB 11:08:47.92 ESA Last receive
   12-FEB 11:08:47.92 ESA Last fork scheduled
   12-FEB 11:08:47.92 ESA Last fork time
   12-FEB 11:08:47.77 ESA5 LAST Last receive
   12-FEB 11:08:47.72 ESA3 LAT Last receive
   12-FEB 11:08:41.25 ESA Last transmit
   12-FEB 11:08:41.25 ESA5 LAST Last transmit
   12-FEB 11:08:40.02 ESA2 DECnet Last receive
   12-FEB 11:08:39.14 ESA2 DECnet Last transmit
   12-FEB 11:08:37.39 ESA3 LAT Last transmit
   12-FEB 10:19:25.31 ESA Last unavail user buffer
   11-FEB 14:10:20.09 ESA5 LAST Last start completed
   11-FEB 14:10:02.16 ESA3 LAT Last start completed
   11-FEB 14:09:58.44 ESA2 DECnet Last start completed
   11-FEB 14:09:57.44 ESA Last DAT transmit

   The **SHOW LAN/TIME** command displays print time information from device and unit data structures.

3. **SDA> SHOW LAN/VCI/DEVICE=ICB**

   -- ICB VCI Information 17-APR-1996 14:22:07 --

   LSB address = 80A1D580
   Device state = 00000003 RUN,INITED

   -- ICB2 80-41 (LAST) VCI Information 17-APR-1996 14:22:07 --

   VCIB address = 8096F238
   CLIENT flags: 00000001 RCV_DCB
   LAN flags: 00000004 LAN_INIT
   DLL flags: 00000005 XMT_CHAIN,PORT_STATUS
   UCB status: 00000015 STRTN,UNIQ,STRTD
   VCI ID LAST VCI version 00010001
   UCB address 80A4C5C0 DP VCRP address 00000000
   Hardware address 00-00-93-08-52-CF LDC address 80A1D720
   Physical address 00-00-93-08-52-CF LAN medium TR
   Transmit available 80A1D670 Outstanding operations 0
   Maximum receives 0 Outstanding receives 0
   Max xmt size 4444 Header size 52
   Build header rtn 808BF230 Report event rtn 86327130
   XMT initiate rtn 808BF200 Transmit complete rtn 86326D80
   XMT frame rtn 808BF210 Receive complete rtn 86326A80

   -- ICB2 80-41 (LAST) VCI Information (cont) 17-APR-1996 14:22:07 --

   Portmgmt initiate rtn 808BF0C0 Portmgmt complete rtn 86327100
   Monitor request rtn 00000000 Monitor transmit rtn 00000000
   Monitor flags 00000000 Monitor receive rtn 00000000
   Port usable 00000000 Port unusable 00000000

   The **SHOW LAN/VCI/DEVICE=ICB** command displays the VCIB for a Token Ring device (ICB) which has an active VCI user (LAST).
The SHOW LAN/ELAN command displays information for the parent ATM device (HCA) driver and the ELAN pseudo-device (ELA) driver.
5. **SDA> SHOW LAN/ELAN/DEV=ELA**

   -- ELA Emulated LAN LSB Information 17-APR-1996 14:08:22 --

   LSB address = 80AB08C0
   Device state = 00000001 RUN

   ELAN name = ELAN 1
   ELAN description = ATM ELAN
   ELAN parent = HCA0
   ELAN state = 00000001 ACTIVE

   MAX transmit size MTU_1516
   LEC attr buff adr 80AB1FC0
   LEC attr buff size 00000328
   Event mask 00000000
   Extended sense 00000000

   -- ELA Emulated LAN LEC Attributes 17-APR-1996 14:08:22 --

   LAN type 00000000
   Proxy flag 00000000
   Max UF count 00000001
   VCC timeout 000004B0
   LEC id 00000002
   Flush timeout 00000004
   SM state 00000070
   CTRL xmt failures 00000000
   CTRL frames_rcvd 00000012
   LERAPS rcvd 00000000
   UCASTs flooded 00000006
   UCASTs sent 00000000
   Local ESI 00000000
   BUS ATM addr 3999990000000008002BA57E80.AA000302FF12.00
   LES ATM addr 3999990000000008002BA57E80.08002B2240A0.00
   My ATM addr 3999990000000008002BA57E80.08002B2240A0.00

   The SHOW LAN/ELAN/DEVICE=ELA command displays information for the
   ELAN pseudo-device (ELA) driver only.

6. **SDA> SHOW LAN/ELAN/DEVICE=HCA**

   -- HCA Emulated LAN LSB Information 17-APR-1996 14:08:25 --

   LSB address = 8098D200
   Device state = 00000101 RUN,RING_AVAL

   Driver CM VC setup adr 808986A0
   Driver CM VC teardown adr 80898668
   NIPG CM handle adr 8096C30C
   NIPG CM SVC handle 00000000
   NIPG CM agent handle adr 809B364C
   NIPG CM mgr lineup handle 809B394C
   NIPG CM IIMI IO handle 809B378C
   MIB II handle adr 809B94CC
   MIB handle adr 809B3ACC
   Queue header for EL LSBs 00000000
   DEC MIB handle adr 809BBD8C
   NIPG current TQEs used 00000000
   Count of allocated TQEs 00000000
   NIPG pool allocations 000757B2

   The SHOW LAN/ELAN/DEVICE=HCA command displays information for the
   ATM device (HCA) driver only.
SHOW LOCKS

Displays information about all lock management locks in the system, or about a specified lock.

Format

SHOW LOCKS {lock-id | /ADDRESS=n | /ALL (d) | /BLOCKING | /BRIEF | /CACHED | /CONVERT | /GRANTED | /NAME=name | /POOL | /STATUS=(keyword [,keyword...]) | /SUMMARY | /WAITING}

Parameter

lock-id
Name of a specific lock.

Qualifiers

/ADDRESS=n
Displays a specific lock, given the address of the lock block.

/ALL
Lists all locks that exist in the system. This is the default behavior of the SHOW LOCK command.

/BLOCKING
Displays only the locks that have a blocking AST specified or attached.

/BRIEF
Displays a single line of information for each lock.

/CACHED
Displays locks that are no longer valid. The memory for these locks is saved so that later requests for locks can use them. Cached locks are not displayed in the other SHOW LOCK commands.

/CONVERT
Displays only the locks that are on the conversion queue.

/GRANTED
Displays only the locks that are on the granted queue.

/NAME=name
Displays a specified lock with the given name.

/POOL
Displays the lock manager’s poolzone information, which contains the lock blocks (LKB) and resource blocks (RSB).

/STATUS=(keyword [,keyword...])
Displays only the locks that have the specified status bits set in the LKB$L_STATUS field. Status keywords are as follows:
### SHOW LOCKS

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2PC_IP</td>
<td>Indicates a two-phase operation in progress</td>
</tr>
<tr>
<td>2PC_PEND</td>
<td>Indicates a two-phase operation pending</td>
</tr>
<tr>
<td>ASYNC</td>
<td>Completes request asynchronously</td>
</tr>
<tr>
<td>BLKASTFLG</td>
<td>Specifies a blocking AST</td>
</tr>
<tr>
<td>BLKASTQED</td>
<td>Indicates a blocking AST is queued</td>
</tr>
<tr>
<td>BRL</td>
<td>Indicates a byte range lock</td>
</tr>
<tr>
<td>CACHED</td>
<td>Indicates a lock block in cache</td>
</tr>
<tr>
<td>CVTSUBRNG</td>
<td>Indicates a sub-range convert request</td>
</tr>
<tr>
<td>CVTTOSYS</td>
<td>Converts back to system-owned lock</td>
</tr>
<tr>
<td>DBLKAST</td>
<td>Delivers a blocking AST</td>
</tr>
<tr>
<td>DCP</td>
<td>Indicates a delete pending cache lock</td>
</tr>
<tr>
<td>FLOCK</td>
<td>Indicates a fork lock</td>
</tr>
<tr>
<td>GRSUBRNG</td>
<td>Grants sub-range lock</td>
</tr>
<tr>
<td>IP</td>
<td>Indicates operation in process</td>
</tr>
<tr>
<td>MSTCPY</td>
<td>Indicates a lock block is a master copy</td>
</tr>
<tr>
<td>NEWSUBRNG</td>
<td>Indicates a new sub-range request</td>
</tr>
<tr>
<td>NOQUOTA</td>
<td>Does not charge quota</td>
</tr>
<tr>
<td>PCACHED</td>
<td>Indicates lock block needs to be cached</td>
</tr>
<tr>
<td>PROTECT</td>
<td>Indicates a protected lock</td>
</tr>
<tr>
<td>RESEND</td>
<td>Resends during failover</td>
</tr>
<tr>
<td>RM_RBRQD</td>
<td>Requires remaster rebuild</td>
</tr>
<tr>
<td>RNGBLK</td>
<td>Specifies a range block</td>
</tr>
<tr>
<td>RNGCHG</td>
<td>Indicates a changing range</td>
</tr>
<tr>
<td>TIMEOUTQ</td>
<td>Indicates lock block is on timeout queue</td>
</tr>
<tr>
<td>VALBLKRD</td>
<td>Indicates read access to lock value block</td>
</tr>
<tr>
<td>VALBLKWRT</td>
<td>Indicates write access to lock value block</td>
</tr>
<tr>
<td>WASSYSOWN</td>
<td>Indicates was system-owned lock</td>
</tr>
</tbody>
</table>

/SUMMARY
Displays summary data and performance counters.

/WAITING
Displays only the waiting locks.

### Description

The SHOW LOCKS command displays the information described in Table 4–4 for each lock management lock in the system, or for the lock indicated by `lock-id`, an address or name. (Use the SHOW SPINLOCKS command to display information about spinlocks.) You can obtain a similar display for the locks owned by a specific process by issuing the appropriate SHOW PROCESS/LOCKS command. See the OpenVMS Programming Concepts Manual for additional information.
You can display information about the resource to which a lock is queued by issuing the SHOW RESOURCES command specifying the resource's lock-id.

Table 4–4 Contents of the SHOW LOCK and SHOW PROCESS/LOCKS Displays

<table>
<thead>
<tr>
<th>Display Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Index¹</td>
<td>Index in the PCB array to a pointer to the process control block (PCB) of the process that owns the lock.</td>
</tr>
<tr>
<td>Name¹</td>
<td>Name of the process that owns the lock.</td>
</tr>
<tr>
<td>Extended PID¹</td>
<td>Clusterwide identification of the process that owns the lock.</td>
</tr>
<tr>
<td>Lock ID</td>
<td>Identification of the lock.</td>
</tr>
<tr>
<td>PID</td>
<td>Systemwide identification of the lock.</td>
</tr>
<tr>
<td>Flags</td>
<td>Information specified in the request for the lock.</td>
</tr>
<tr>
<td>Par. ID</td>
<td>Identification of the lock's parent lock.</td>
</tr>
<tr>
<td>Sublocks</td>
<td>Count of the locks that the lock owns.</td>
</tr>
<tr>
<td>LKB</td>
<td>Address of the lock block (LKB). If a blocking AST has been enabled for this lock, the notation “BLKAST” appears next to the LKB address.</td>
</tr>
<tr>
<td>Priority</td>
<td>The lock priority.</td>
</tr>
<tr>
<td>Granted at</td>
<td>Lock mode at which the lock was granted.</td>
</tr>
<tr>
<td>RSB</td>
<td>Address of the resource block.</td>
</tr>
<tr>
<td>Resource</td>
<td>Dump of the resource name. The two leftmost columns of the dump show its contents as hexadecimal values, the least significant byte being represented by the rightmost two digits. The rightmost column represents its contents as ASCII text, the least significant byte being represented by the leftmost character.</td>
</tr>
<tr>
<td>Status</td>
<td>Status of the lock, information used internally by the lock manager.</td>
</tr>
<tr>
<td>Length</td>
<td>Length of the resource name.</td>
</tr>
<tr>
<td>Mode</td>
<td>Processor access mode of the namespace in which the resource block (RSB) associated with the lock resides.</td>
</tr>
<tr>
<td>Owner</td>
<td>Owner of the resource. Certain resources owned by the operating system list “System” as the owner. Resources owned by a group have the number (in octal) of the owning group in this field.</td>
</tr>
<tr>
<td>Copy</td>
<td>Indication of whether the lock is mastered on the local system or is a process copy.</td>
</tr>
</tbody>
</table>

¹This display element is produced only by the SHOW PROCESS/LOCKS command.
### Examples

1. **SDA> SHOW LOCKS**

   Lock Database
   ---------------
   
   Lock id: 3E000002  PID: 00000000  Flags: CONVERT NOQUEUE SYNCSTS
   Par. id: 00000000  SUBLCKs: 0  NOQUOTA CVTSYS
   LKB: FFFFFFFF.7DF48150  BLKAST: 81107278
   Priority: 0000
   Granted at CR 00000000-FFFFFFFF
   RSB: FFFFFFFF.7DF68D50
   Resource: 494D6224 42313146  F11B$sMI  Status: NOQUOTA VALBLKR VALBLKW
   Length 18 4D55445F 5944414C  LADY_DUM
   System 00000000 00000000 ........
   Local copy

   Lock Database
   ---------------
   
   Lock id: 3F000003  PID: 00000000  Flags: VALBLK CONVERT SYNCSTS
   Par. id: 0100007A  SUBLCKs: 0  CVTSYS
   LKB: FFFFFFFF.7DF48250  BLKAST: 00000000
   Priority: 0000
   Granted at NL 00000000-FFFFFFFF
   RSB: FFFFFFFF.7DF51D50
   Resource: 01F77324 42313146  F11B$s÷. Status: NOQUOTA VALBLKR VALBLKW
   Length 10 00000000 00000000 ........
   System 00000000 00000000 ........
   Local copy

   Lock Database
   ---------------
   
   Lock id: 0A000004  PID: 0001000F  Flags: VALBLK CONVERT SYNCSTS
   Par. id: 00000000  SUBLCKs: 0  SYSTEM NODLCKW NODLCKB
   LKB: FFFFFFFF.7DF48350  BLKAST: 81190420  QUECVT
   Priority: 0000
   Granted at EX 00000000-FFFFFFFF
   RSB: FFFFFFFF.7DF50850
   Resource: 004F0DF2 24534D52  RMS$ß.O. Status: VALBLKR VALBLKW
   Length 26 5F313039 58020000 ...X901_
   Exec. mode 00202020 204C354B  K5L .
   System 00000000 00000000 ........
   Local copy
   ...
   ...

---

**SDA Commands SHOW LOCKS**
SDA Commands

SHOW LOCKS

2. SDA> SHOW RESOURCES/LOCKID=0A000004

Resource Database

-------------------

RSB: FFFFFFFF.7DF50850 GGMODE: EX Status: DIRENTR VALID
Parent RSB: 00000000.00000000 CGMODE: EX
Sub-RSB count: 0 FGMODE: EX
Lock Count: 1 RQSEQNM: 0000
BLKAST count: 1 CSID: 00000000 (MILADY)

Resource: 004F00DF HFS$A.O. Valblk: 00000000 00000000
Length 26 5F313039 58020000 ...X901_ 00000000 00000000
Exec. mode 00202020 204C354B KSL
System: 00000000 00000000

Granted queue (Lock ID / Gr mode / Range):
0A000004 EX 00000000-FFFFFFFF

Conversion queue (Lock ID / Gr mode / Range -> Rq mode / Range):
*** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode / Range):
*** EMPTY QUEUE ***

This SDA session shows the output of the SHOW LOCKS command for several locks. The SHOW RESOURCES command, executed for the last displayed lock, verifies that the lock is in the resource's granted queue. (See Table 4–21 for a full explanation of the contents of the display of the SHOW RESOURCES command.)

3. SDA> SHOW LOCK/BRIEF/BLOCKING

Lock Database

<table>
<thead>
<tr>
<th>LKB Address</th>
<th>Lockid</th>
<th>ParentId</th>
<th>PID</th>
<th>BLKAST</th>
<th>SubLocks</th>
<th>RQ</th>
<th>GR</th>
<th>Queue</th>
<th>RSB Address</th>
<th>Resource Name</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFF.7DF42450</td>
<td>51000003</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>80CC7648</td>
<td>0</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF45050</td>
<td>F11B$sCAPTURE_DUMP</td>
<td>Kern</td>
</tr>
<tr>
<td>FFFFFFFF.7DF42850</td>
<td>01000005</td>
<td>00000000</td>
<td>00000000</td>
<td>00CB5020</td>
<td>111</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF42950</td>
<td>F11B$sX6JU_R3N</td>
<td>Kern</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF42A50</td>
<td>01000006</td>
<td>00000000</td>
<td>00000000</td>
<td>80CD3D98</td>
<td>0</td>
<td>PR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF42B50</td>
<td>F11B$sX6JU_R3N</td>
<td>Kern</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF42E50</td>
<td>4D000008</td>
<td>00000000</td>
<td>00000000</td>
<td>80CC7648</td>
<td>0</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF43150</td>
<td>F11B$sX6JU_R3N</td>
<td>Kern</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF43E50</td>
<td>13000010</td>
<td>00000000</td>
<td>00000000</td>
<td>80CD3D98</td>
<td>0</td>
<td>PR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF48550</td>
<td>APPENDER</td>
<td>Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF49550</td>
<td>1500003A</td>
<td>00000000</td>
<td>00010008</td>
<td>00010B20</td>
<td>0</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF54850</td>
<td>AUDRSV$SQL...X6JU_R3N</td>
<td>... User</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF49B50</td>
<td>0100003D</td>
<td>00000000</td>
<td>00010007</td>
<td>00035EF8</td>
<td>0</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF56250</td>
<td>OPC$opcom-restart</td>
<td>User</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4BE50</td>
<td>2100004F</td>
<td>00000000</td>
<td>0001000B</td>
<td>80CE66F0</td>
<td>4</td>
<td>NL</td>
<td>Granted</td>
<td>FFFFFFFF.7DF4DC50</td>
<td>RMS$y......X6JU_R3N</td>
<td>... Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4C950</td>
<td>12000033</td>
<td>03000094</td>
<td>00010008</td>
<td>80CE7220</td>
<td>0</td>
<td>PW</td>
<td>Granted</td>
<td>FFFFFFFF.7DF48E50</td>
<td>APPENDER</td>
<td>Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF49550</td>
<td>1500003A</td>
<td>00000000</td>
<td>00010008</td>
<td>00010B20</td>
<td>0</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF54850</td>
<td>AUDRSV$SQL...X6JU_R3N</td>
<td>... User</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF49B50</td>
<td>0100003D</td>
<td>00000000</td>
<td>00010007</td>
<td>00035EF8</td>
<td>0</td>
<td>CR</td>
<td>Granted</td>
<td>FFFFFFFF.7DF56250</td>
<td>OPC$opcom-abort</td>
<td>User</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4BE50</td>
<td>2100004F</td>
<td>00000000</td>
<td>0001000B</td>
<td>80CE66F0</td>
<td>4</td>
<td>NL</td>
<td>Granted</td>
<td>FFFFFFFF.7DF4DC50</td>
<td>RMS$y......X6JU_R3N</td>
<td>... Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4C950</td>
<td>12000033</td>
<td>03000094</td>
<td>00010008</td>
<td>80CE7220</td>
<td>0</td>
<td>PW</td>
<td>Granted</td>
<td>FFFFFFFF.7DF48E50</td>
<td>APPENDER</td>
<td>Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4E050</td>
<td>0B00005F</td>
<td>00000000</td>
<td>00010009</td>
<td>80CE66F0</td>
<td>4</td>
<td>NL</td>
<td>Granted</td>
<td>FFFFFFFF.7DF4AD50</td>
<td>RMS$y......X6JU_R3N</td>
<td>... Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4E050</td>
<td>0B00005F</td>
<td>00000000</td>
<td>00010009</td>
<td>80CE66F0</td>
<td>4</td>
<td>NL</td>
<td>Granted</td>
<td>FFFFFFFF.7DF4AD50</td>
<td>RMS$y......X6JU_R3N</td>
<td>... Exec</td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.7DF4E050</td>
<td>0B00005F</td>
<td>00000000</td>
<td>00010009</td>
<td>80CE66F0</td>
<td>4</td>
<td>NL</td>
<td>Granted</td>
<td>FFFFFFFF.7DF4AD50</td>
<td>RMS$y......X6JU_R3N</td>
<td>... Exec</td>
<td></td>
</tr>
</tbody>
</table>

This example shows the brief display for all locks with a blocking AST.
SHOW MACHINE_CHECK

Displays the contents of the stored machine check frame. This command is valid for the DEC 4000 Alpha, DEC 7000 Alpha, and DEC 10000 Alpha computers only.

Format

SHOW MACHINE_CHECK [/FULL] [cpu-id]

Parameter

cpu-id
Numeric value from 00 to 1F_{16} indicating the identity of the CPU for which context information is to be displayed. This parameter changes the SDA current CPU (the default) to the CPU specified with cpu-id. If you specify a value outside this range, or you specify the cpu-id of a processor that was not active at the time of the system failure, SDA displays the following message:

%SDA-E-CPUNOTVLD, CPU not booted or CPU number out of range

If you use the cpu-id parameter, the SHOW MACHINE_CHECK command performs an implicit SET CPU command, making the CPU indicated by cpu-id the current CPU for subsequent SDA commands. (See the description of the SET CPU command and Chapter 2, Section 2.5 for information on how this can affect the CPU context—and process context—in which SDA commands execute.)

Qualifier

/FULL
Specifies that a detailed version of the machine check information be displayed. This is currently identical to the default summary display.

Description

The SHOW MACHINE_CHECK command displays the contents of the stored machine check frame. A separate frame is allocated at boot time for every CPU in a multiple-CPU system. This command is valid for the DEC 4000 Alpha, DEC 7000 Alpha, and DEC 10000 Alpha computers only.

If you do not specify a qualifier, a summary version of the machine check frame is displayed.

The default cpu-id is the SDA current CPU.
Examples

1. SDA> SHOW MACHINE_CHECK
CPU 00 Stored Machine Check Crash Data
--------------------------------------

Processor specific information:

Exception address: FFFFFFFF.800B0250 Exception Summary: 00000000.00000000
Pal base address: 00000000.00000800 Exception Mask: 00000000.00000000
HW Interrupt Request: 00000000.00000342 HW Interrupt Ena: 00000001.FF01CE0
MM_CSR: 00000000.000003640 ICSR: 00000002.381F0000
D-cache address: 00000000.FFFFFFFF D-cache status: 00000000.000002E0
BIU status: 00000000.00000050 BIU address [7..0]: 00000000.000060E0
BIU control: 00000000.50006447 Fill Address: 00000000.000006120
Single-bit syndrome: 00000000.00000000 Processor mchck VA: 00000000.00006190
A-box control: 00000000.0000040E B-cache TAG: 00106100.83008828

System specific information:

Garbage bus info: 00200009 00000038 Device type: 000B8001
LCNR: 00000001 Memory error: 00000000
LBER: 00000009 Bus error synd 0,1: 00000000 00000000
Bus error cmd: 00048858 00AB1C88 Bus error synd 2,3: 00000000 0000002C
LEP mode: 00010010 LEP lock address: 00041108

The SHOW MACHINE_CHECK command in this SDA display shows the contents of the stored machine check frame.

2. SDA> SHOW MACHINE_CHECK 1
CPU 01 Stored Machine Check Crash Data
--------------------------------------

Processor specific information:

Exception address: FFFFFFFF.800868A0 Exception Summary: 00000000.00000000
Pal base address: 00000000.00000800 Exception Mask: 00000000.00000000
HW Interrupt Request: 00000000.00000342 HW Interrupt Ena: 00000001.FF01CE0
MM_CSR: 00000000.000003640 ICSR: 00000002.381F0000
D-cache address: 00000000.FFFFFFFF D-cache status: 00000000.000002E0
BIU status: 00000000.00000050 BIU address [7..0]: 00000000.000060E0
BIU control: 00000000.50006447 Fill Address: 00000000.000006120
Single-bit syndrome: 00000000.00000000 Processor mchck VA: 00000000.00006190
A-box control: 00000000.0000040E B-cache TAG: 35028EA0.50833828

System specific information:

Garbage bus info: 00210001 00000038 Device type: 000B8001
LCNR: 00000001 Memory error: 00000000
LBER: 00040209 Bus error synd 0,1: 00000000 00000000
Bus error cmd: 00048858 00AB1C88 Bus error synd 2,3: 00000000 0000002C
LEP mode: 00010010 LEP lock address: 00041108

The SHOW MACHINE_CHECK command in this SDA display shows the contents of the stored machine check frame for cpu-id 01.
SHOW MEMORY

Displays the availability and usage of memory resources.

Format

SHOW MEMORY [/ALL][/BUFFER_OBJECTS][/CACHE][/FILES]
[/FULL][/GH_REGIONS][/PHYSICAL_PAGES][/POOL]
[/RESERVED][/SLOTS]

Parameters

None.

Qualifiers

/ALL
Displays all available information, that is, information displayed by the following qualifiers:

/BUFFER_OBJECTS
/CACHE
/FILES
/GH_REGIONS
/PHYSICAL_PAGES
/POOL
/RESERVED
/SLOTS

This is the default display.

/BUFFER_OBJECTS
Displays information about system resources used by buffer objects.

/CACHE
Displays information about either the Virtual I/O Cache facility, or the Extended File Cache facility. The system parameter VCC_FLAGS determines which is used. The cache facility information is displayed as part of the SHOW MEMORY and SHOW MEMORY/CACHE/FULL commands.

/FILES
Displays information about the use of each paging and swapping file currently installed.

/FULL
Displays additional information about each pool area or paging or swapping file currently installed, when used with the /POOL or the /FILES qualifier. This qualifier is ignored unless the /FILES or the /POOL qualifier is specified explicitly. When used with the /CACHE qualifier, /FULL displays additional information about the use of the Virtual I/O Cache facility, but is ignored if the Extended File Cache facility is in use.

/GH_REGIONS
Displays information about the granularity hint regions (GHR) that have been established. For each of these regions, information is displayed about the size of the region, the amount of free memory, the amount of memory in use, and
SDA Commands
SHOW MEMORY

the amount of memory released to OpenVMS from the region. The granularity hint regions information is also displayed as part of SHOW MEMORY, SHOW MEMORY/ALL, and SHOW MEMORY/FULL commands.

/PHYSICAL_PAGES
Displays information about the amount of physical memory and the number of free and modified pages.

/POOL
Displays information about the usage of each dynamic memory (pool) area, including the amount of free space and the size of the largest contiguous block in each area.

/RESERVED
Displays information about memory reservations.

/SLOTS
Displays information about the availability of partition control block (PCB) vector slots and balance slots.

Description

For more details on the SHOW MEMORY command, see the description in OpenVMS DCL Dictionary: N-Z.
SHOW PAGE_TABLE

Displays a range of system page table entries, the entire system page table, or the entire global page table.

Format

SHOW PAGE_TABLE  {range | /FREE [/HEADER=address] | /GLOBAL | /GPT | /PT | /INVALID_PFN [=option] | /NONMEMORY_PFN [=option] | /PTE_ADDRESS | /SECTION_INDEX=n | /S0S1 (d) | /S2 | /SPTW | =ALL} {/L1 | /L2 | /L3 (d)}

Parameter

range
Range of virtual addresses or PTE addresses for which SDA displays page table entries. If the qualifier /PTE_ADDRESS is given, then the range is of PTE addresses; otherwise, the range is of virtual addresses.

If /PTE_ADDRESS is given, the range is expressed using the following syntax:

m Displays the single page table entry at address m
m:n Displays the page table entries from address m to address n
m;n Displays n bytes of page table entries starting at address m

If /PTE_ADDRESS is not given, then range is expressed using the following syntax:

m Displays the single page table entry that corresponds to virtual address m
m:n Displays the page table entries that correspond to the range of virtual addresses from m to n
m;n Displays the page table entries that correspond to a range of n bytes starting at virtual address m

Qualifiers

/FREE
Causes the starting addresses and sizes of blocks of pages in the free page list to be displayed. The qualifiers /S0S1 (default), /S2, /GLOBAL, and /HEADER determine which free page list is to be displayed.

/GLOBAL
Lists the global page table. When used with the /FREE qualifier, /GLOBAL indicates the free page list to be displayed.

/HEADER=address
When used with the /FREE qualifier, the /HEADER=address qualifier displays the free list for the specified private page table.

/GPT
Specifies the portion of page table space that maps the global page table as the address range.
/INVALID_PFN [option]
The /INVALID_PFN qualifier, which is valid on platforms that supply an I/O memory map, causes SDA to display only page table entries that map to PFNs that are not in the system's private memory, nor in Galaxy shared memory, nor are I/O access pages.

See the /NONMEMORY_PFN qualifier definition for a description of the options.

/L1
Lists the Level 1 page table entries for the portion of memory specified.

/L2
Lists the Level 2 page table entries for the portion of memory specified.

/L3
Lists the Level 3 page table entries for the portion of memory specified. This qualifier is the default level.

/NONMEMORY_PFN [option]
The /NONMEMORY_PFN qualifier, supported on all platforms, causes SDA to display only page table entries that are neither in the system's private memory nor in Galaxy shared memory.

Both /INVALID_PFN and /NONMEMORY_PFN qualifiers allow two optional keywords, READONLY and WRITABLE. If neither keyword is given, all relevant pages are displayed. If READONLY is given, only pages marked for no write access are displayed. If WRITABLE is given, only pages that allow write access are displayed. For example, SHOW PAGE_TABLE=ALL/INVALID_PFN=WRITABLE would display all system pages whose protection allows write, but which map to PFNs that do not belong to this system.

/PT
Specifies page table space, as viewed from system context, as the address range.

/PTE_ADDRESS
Specifies that the range given is of PTE addresses instead of the virtual addresses mapped by the PTEs.

/SECTION_INDEX=n
Displays the page table for the range of pages in the global section or pageable part of a loaded image. For pageable portions of loaded images, one of the qualifiers /L1, /L2, or /L3 can also be specified.

/S0S1
Specifies S0 and S1 space as the address range. When used with the /FREE qualifier, /S0S1 indicates the free page list to be displayed. This is the default portion of memory or free page list to be displayed.

/S2
Specifies S2 space as the address range. When used with the /FREE qualifier, /S2 indicates the free page list to be displayed.

/SPTW
Displays the contents of the system page table window.
### Option

**=ALL**
The SHOW PAGE = ALL command displays the page table entries for all shared (system) addresses, without regard to the section of memory being referenced. It is equivalent to specifying all of /S0S1, /S2, /SPTW, /PT, /GPT, and /GLOBAL. This option can be qualified by only one of the /L1, /L2, or /L3 qualifiers.

### Description

If the /FREE qualifier is not specified, this command displays page table entries for the specified range of addresses or section of memory. For each virtual address displayed by the SHOW PAGE_TABLE command, the first eight columns of the listing provide the associated page table entry and describe its location, characteristics, and contents. SDA obtains this information from the system page table. Table 4–5 describes the information displayed by the SHOW PAGE_TABLE command.

If the /FREE qualifier is specified, this command displays the free PTE list for the specified section of memory.

Note that the /L1, /L2, and /L3 qualifiers are ignored when used with the /FREE, /GLOBAL, and /SPTW qualifiers.

### Table 4–5 Virtual Page Information in the SHOW PAGE_TABLE Display

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPPED ADDRESS</td>
<td>Virtual address that marks the base of the virtual page(s) mapped by the PTE.</td>
</tr>
<tr>
<td>PTE ADDRESS</td>
<td>Virtual address of the page table entry that maps the virtual page(s).</td>
</tr>
<tr>
<td>PTE</td>
<td>Contents of the page table entry, a quadword that describes a system virtual page.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Type of virtual page. Table 4–6 shows the eight types and their meanings.</td>
</tr>
<tr>
<td>READ</td>
<td>A code, derived from bits in the PTE, that designates the processor access modes (kernel, executive, supervisor, or user) for which read access is granted.</td>
</tr>
<tr>
<td>WRIT</td>
<td>A code, derived from bits in the PTE, that designates the processor access modes (kernel, executive, supervisor, or user) for which write access is granted.</td>
</tr>
<tr>
<td>BITS</td>
<td>Letters that represent the setting of a bit or a combination of bits in the PTE. These bits indicate attributes of a page. Table 4–7 shows the codes and their meanings.</td>
</tr>
<tr>
<td>GH</td>
<td>Contents of granularity hint bits.</td>
</tr>
</tbody>
</table>
Table 4–6  Type of Virtual Pages

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALID</td>
<td>Valid page (in main memory).</td>
</tr>
<tr>
<td>TRANS</td>
<td>Transitional page (on free or modified page list).</td>
</tr>
<tr>
<td>DZERO</td>
<td>Demand-allocated, zero-filled page.</td>
</tr>
<tr>
<td>PGFIL</td>
<td>Page within a paging file.</td>
</tr>
<tr>
<td>STX</td>
<td>Section table's index page.</td>
</tr>
<tr>
<td>GPTX</td>
<td>Index page for a global page table.</td>
</tr>
<tr>
<td>IOPAG</td>
<td>Page in I/O address space.</td>
</tr>
<tr>
<td>NXMEM</td>
<td>Page not represented in physical memory. The page frame number (PFN) of this page is not mapped by any of the system’s memory controllers. This indicates an error condition.</td>
</tr>
</tbody>
</table>

Table 4–7  Bits In the PTE

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Address space match is set.</td>
</tr>
<tr>
<td>M</td>
<td>Page has been modified.</td>
</tr>
<tr>
<td>L</td>
<td>Page is locked into a working set.</td>
</tr>
<tr>
<td>K</td>
<td>Owner is kernel mode.</td>
</tr>
<tr>
<td>E</td>
<td>Owner is executive mode.</td>
</tr>
<tr>
<td>S</td>
<td>Owner is supervisor mode.</td>
</tr>
<tr>
<td>U</td>
<td>Owner is user mode.</td>
</tr>
</tbody>
</table>

If the virtual page has been mapped to a physical page, the last six columns of the listing include information from the page frame number (PFN) database; otherwise, the section is left blank. Table 4–8 describes the physical page information displayed by the SHOW PAGE_TABLE command.

Table 4–8  Physical Page Information in the SHOW PAGE_TABLE Display

<table>
<thead>
<tr>
<th>Category</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGTYP</td>
<td>Type of physical page. Table 4–9 shows the types of physical pages.</td>
</tr>
<tr>
<td>LOC</td>
<td>Location of the page within the system. Table 4–10 shows the possible locations with their meaning.</td>
</tr>
<tr>
<td>BAK</td>
<td>Place to find information on this page when all links to this PTE are broken: either an index into a process section table or the number of a virtual block in the paging file.</td>
</tr>
<tr>
<td>REFCNT</td>
<td>Number of references being made to this page.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 4–8 (Cont.) Physical Page Information in the SHOW PAGE_TABLE Display

<table>
<thead>
<tr>
<th>Category</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLINK</td>
<td>Forward link within PFN database that points to the next physical page (if the page is on one of the lists: FREE, MODIFIED, BAD, or ZEROED); this longword also acts as the count of the number of processes that are sharing this global section.</td>
</tr>
<tr>
<td>BLINK</td>
<td>Backward link within PFN database (if the page is on one of the lists: FREE, MODIFIED, BAD, or ZEROED); also acts as an index into the working set list.</td>
</tr>
</tbody>
</table>

### Table 4–9 Types of Physical Pages

<table>
<thead>
<tr>
<th>Page Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS</td>
<td>Page is part of process space.</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Page is part of system space.</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>Page is part of a global section.</td>
</tr>
<tr>
<td>PPGTBL</td>
<td>Page is part of a process page table.</td>
</tr>
<tr>
<td>PHD&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Page is part of a process PHD.</td>
</tr>
<tr>
<td>PPT(Ln)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Page is a process page table page at level n.</td>
</tr>
<tr>
<td>SPT(Ln)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Page is a system page table page at level n.</td>
</tr>
<tr>
<td>GPGTBL</td>
<td>Page is part of a global page table.</td>
</tr>
<tr>
<td>GBLWRT</td>
<td>Page is part of a global, writable section.</td>
</tr>
<tr>
<td>SHPT&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Page is part of a shared page table.</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown.</td>
</tr>
</tbody>
</table>

<sup>1</sup>These page types are variants of the PPGTBL page type.

<sup>2</sup>The SHPT page type is a variant of the GBLWRT page type.

### Table 4–10 Location of the Page

<table>
<thead>
<tr>
<th>Location</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE</td>
<td>Page is in a working set.</td>
</tr>
<tr>
<td>MFYLST</td>
<td>Page is in the modified page list.</td>
</tr>
<tr>
<td>FRELST</td>
<td>Page is in the free page list.</td>
</tr>
<tr>
<td>BADLST</td>
<td>Page is in the bad page list.</td>
</tr>
<tr>
<td>RELPND</td>
<td>Release of the page is pending.</td>
</tr>
<tr>
<td>RDERR</td>
<td>Page has had an error during an attempted read operation.</td>
</tr>
<tr>
<td>PAGOUT</td>
<td>Page is being written into a paging file.</td>
</tr>
<tr>
<td>PAGIN</td>
<td>Page is being brought into memory from a paging file.</td>
</tr>
<tr>
<td>ZROLST</td>
<td>Page is in the zeroed-page list.</td>
</tr>
<tr>
<td>UNKNWN</td>
<td>Location of page is unknown.</td>
</tr>
</tbody>
</table>

SDA indicates pages are inaccessible by displaying one of the following messages:
In this case, the page table entries are not in use (page referenced is inaccessible).

In this case, the page table entries do not exist (PTE itself is inaccessible).

In this case, the page table entries are in the list of free system pages.

In each case, “VA” is the MAPPED ADDRESS of the skipped entry, and “PTE” is the PTE ADDRESS of the skipped entry.

**Examples**

1. For an example of SHOW PAGE_TABLE output when the qualifier /FREE has not been given, see the SHOW PROCESS/PAGE_TABLES command.

2. SDA> SHOW PAGE_TABLE/FREE

   S0/S1 Space Free PTEs

   ------------------------------
   MAPPED ADDRESS   PTE ADDRESS   PTE   COUNT
   FFFFFFFF.82A08000  FFFFFFFF.FFE0A820  0001FFE0.A8580000  00000003
   FFFFFFFF.82A16000  FFFFFFFF.FFE0A858  0001FFE0.A8900000  00000003
   FFFFFFFF.82A24000  FFFFFFFF.FFE0A890  0001FFE0.B3C00000  00000003
   FFFFFFFF.82CF0000  FFFFFFFF.FFE0B3C0  0001FFE0.B4010000  00000001
   FFFFFFFF.82D00000  FFFFFFFF.FFE0B400  0001FFE0.B4680000  00000002
   .
   .
   .
   FFFFFFFF.82E48000  FFFFFFFF.FFE0B920  0001FFE0.B9390000  00000001
   FFFFFFFF.82E4E000  FFFFFFFF.FFE0B938  0001FFE0.BA200000  00000002
   FFFFFFFF.82E88000  FFFFFFFF.FFE0B9A0  0001FFE0.C97B0000  00000003
   FFFFFFFF.8325E000  FFFFFFFF.FFE0CC978  0001FFE0.CC9B0000  00000003
   FFFFFFFF.8332B000  FFFFFFFF.FFE0C930  00000000.00000000  00000006D

   This example shows the output when you invoke the SHOW PAGE_TABLE/FREE command.
SHOW PARAMETER

Displays the name, location, and value of one or more SYSGEN parameters at the time that the system dump is taken.

Format

SHOW PARAMETER  [SYSGEN_parameter]
  [/ACP][/ALL][/CLUSTER][/DYNAMIC][/GALAXY]
  [/GEN][/JOB][/LGI][/MAJOR][/MULTIPROCESSING]
  [/PQL][/RMS][/SCS][/SPECIAL][/SYS][/STARTUP]
  [/TTY]

Parameter

SYSGEN_parameter
The name of a parameter to be displayed. The name given may include wildcards. However, a truncated name is not recognized, unlike the equivalent SYSGEN and SYSMAN commands.

Qualifiers

/ACP
Displays all Files-11 ACP parameters.

/ALL
Displays the values of all parameters except the special control parameters.

/CLUSTER
Displays all parameters specific to clusters.

/DYNAMIC
Displays all parameters that can be changed on a running system.

/GALAXY
Displays all parameters specific to Galaxy systems.

/GEN
Displays all general parameters.

/JOB
Displays all Job Controller parameters.

/LGI
Displays all LOGIN security control parameters.

/MAJOR
Displays the most important parameters.

/MULTIPROCESSING
Displays parameters specific to multiprocessing.

/PQL
Displays the parameters for all default and minimum process quotas.
SDA Commands
SHOW PARAMETER

/RMS
Displays all parameters specific to OpenVMS Record Management Services (RMS).

/SCS
Displays all parameters specific to OpenVMS Cluster System Communications Services.

/SPECIAL
Displays all special control parameters.

/STARTUP
Displays the name of the site-independent startup procedure.

/SYS
Displays all active system parameters.

/TTY
Displays all parameters for terminal drivers.

Description
The SHOW PARAMETER command displays the name, location and value of one or more SYSGEN parameters at the time that the system dump is taken. You can specify either a parameter name, or one or more qualifiers, but not both a parameter and qualifiers. If you do not specify a parameter nor qualifiers, then the last parameter displayed is displayed again.

The qualifiers are the equivalent to those available for the SHOW [parameter] command in the SYSGEN utility and the PARAMETERS SHOW command in the SYSMAN utility. See the OpenVMS System Management Utilities Reference Manual: M–Z for more information about these two commands. You can combine qualifiers, and all appropriate SYSGEN parameters are displayed.

Note
To see the entire set of parameters, use the SDA command SHOW PARAMETER /ALL /SPECIAL /STARTUP.
Examples

1. SDA> SHOW PARAMETER *SCS*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable</th>
<th>Address</th>
<th>Value</th>
<th>(decimal)</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCDBUFFCNT</td>
<td>SCS$GW_BDTCNT</td>
<td>80C159A0</td>
<td>0032</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>SCDCONNCT</td>
<td>SCS$GW_CDTCNT</td>
<td>80C159A8</td>
<td>0005</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>SCDSERVICECT</td>
<td>SCS$GW_BDTCNT</td>
<td>80C159B0</td>
<td>012C</td>
<td>303</td>
<td></td>
</tr>
<tr>
<td>SCDSMAXD</td>
<td>SCS$GW_MAXD</td>
<td>80C159B8</td>
<td>0240</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SCDSMAXMSG</td>
<td>SCS$GW_MAXMSG</td>
<td>80C159C0</td>
<td>0008</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>SCDSFLOWCNT</td>
<td>SCS$GW_FLOWCNT</td>
<td>80C159C8</td>
<td>0001</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SCS$SYSTEMID</td>
<td>SCS$GB_SYSTEMID</td>
<td>80C159D0</td>
<td>000FE88</td>
<td>65160</td>
<td></td>
</tr>
<tr>
<td>SCS$GB_SYSTEMIDH</td>
<td>SCS$GB_SYSTEMIDH</td>
<td>80C159D8</td>
<td>00000000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SCS$NODE</td>
<td>SCS$GB_NODENAME</td>
<td>80C159E0</td>
<td>&quot;SWPCTX  &quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NISCS_CONV_BOOT</td>
<td>CLU$GL_SGN_FLAGS</td>
<td>80C15E68</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NISCS_LOAD_PEA0</td>
<td>CLU$GL_SGN_FLAGS</td>
<td>80C15E68</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>NISCS_PORT_SERV CLU$GL_SNC_PORT_SERV</td>
<td>80C15E70</td>
<td>00000000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC$ICLUST$1</td>
<td>SGN$GL_SCSICLUSTER_P1</td>
<td>80C15E88</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SC$ICLUST$2</td>
<td>SGN$GL_SCSICLUSTER_P2</td>
<td>80C15E90</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SC$ICLUST$3</td>
<td>SGN$GL_SCSICLUSTER_P3</td>
<td>80C15EA8</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SC$ICLUST$4</td>
<td>SGN$GL_SCSICLUSTER_P4</td>
<td>80C15EB0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SC$MAX_PKTSZ</td>
<td>CLU$GL_SCS$MAX_PKTSZ</td>
<td>80C16070</td>
<td>00000000</td>
<td>1498</td>
<td></td>
</tr>
<tr>
<td>SC$LAN_OVRHD</td>
<td>CLU$GL_SCS$LAN_OVRHD</td>
<td>80C16078</td>
<td>00000072</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

This example shows all parameters that have the string "SCS" in their name. Note that for parameters defined as a single bit, the name and value of the bit offset within the location used for the parameter are also given.

2. SDA> SHOW PARAMETER WS*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable</th>
<th>Address</th>
<th>Value</th>
<th>(decimal)</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS$MAX</td>
<td>SGN$GL_MAXWSCNT_PAGELETS</td>
<td>80C15710</td>
<td>00000680</td>
<td>26624</td>
<td></td>
</tr>
<tr>
<td>WS$INC</td>
<td>SGN$GL_MAXWSCNT_PAGES</td>
<td>80C15718</td>
<td>00000068</td>
<td>1664</td>
<td></td>
</tr>
<tr>
<td>WS$DEC</td>
<td>SCH$GL_WSINC_PAGELETS</td>
<td>80C157F8</td>
<td>00000096</td>
<td>2400</td>
<td></td>
</tr>
<tr>
<td>WS$DEC</td>
<td>SCH$GL_WSDEC_PAGELETS</td>
<td>80C15800</td>
<td>00000096</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>WS$INC</td>
<td>SCH$GL_WSINC_PAGES</td>
<td>80C15808</td>
<td>00000096</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>WS$DEC</td>
<td>SCH$GL_WSDEC_PAGELETS</td>
<td>80C15810</td>
<td>00000960</td>
<td>4000</td>
<td></td>
</tr>
</tbody>
</table>

This example shows all parameters whose names begin with the string "WS". Note that for parameters that have both an external value (pagelets) and an internal value (pages), both are displayed.

3. SDA> SHOW PARAMETER /MULTIPROCESSING /STARTUP

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable</th>
<th>Address</th>
<th>Value</th>
<th>(decimal)</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP$CPUS</td>
<td>SGN$GL_SMP_CPUS</td>
<td>80C15688</td>
<td>FFFFFFPF</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>MULTIPROCESSING</td>
<td>SGN$GL_MULTIPROCESSING</td>
<td>80C15698</td>
<td>03</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SMP_SANITY_CNT</td>
<td>SGN$GL_SMP_SANITY_CNT</td>
<td>80C156A8</td>
<td>0000012C</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>SMP_SPINWAIT</td>
<td>SGN$GL_SMP_SPINWAIT</td>
<td>80C156B8</td>
<td>000184A0</td>
<td>100000</td>
<td></td>
</tr>
<tr>
<td>SMP$NSPINWAIT</td>
<td>SGN$GL_SNSPINWAIT</td>
<td>80C156C0</td>
<td>002D62C0</td>
<td>300000</td>
<td></td>
</tr>
<tr>
<td>IO$PREFER_CPUS</td>
<td>SGN$GL_AVAILABLE_PORT_CPUS</td>
<td>80C16130</td>
<td>FFFFFFPF</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

This example shows all the parameters specific to multiprocessing, plus the name of the site-independent startup command procedure.
SHOW PFN_DATA

Displays information that is contained in the page lists and PFN database.

Format

SHOW PFN_DATA {[/qualifier] | pfn [[:end-pfn | ;length]]}

or

SHOW PFN_DATA/MAP

Parameters

pfn
Specifies the page frame number (PFN) of the physical page for which information is to be displayed.

length
Specifies the length of the PFN list to be displayed. When you specify the length parameter, a range of PFNs is displayed. This range starts at the PFN specified by the pfn parameter and contains the number of entries specified by the length parameter.

end-pfn
Specifies the last PFN to be displayed. When you specify the end-pfn parameter, a range of PFNs is displayed. This range starts at the PFN specified by the pfn parameter and ends with the PFN specified by the end-pfn parameter.

Qualifiers

/ADDRESS=<PFN-entry-address>
Displays the PFN database entry at the address specified. The address specified is rounded to the nearest entry address so if you have an address that points to one of the fields of the entry, the correct database entry will still be found.

/ALL
Displays the following lists:

  free page list
  zeroed free page list
  modified page list
  bad page list
  untested page list
  private page lists, if any
  per-color free and zeroed free page lists
  entire database in order by page frame number

This is the default behavior of the SHOW PFN_DATA command. SDA precedes each list with a count of the pages it contains and its low and high limits.

/BAD
Displays the bad page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.
/COLOR \([=\(n\mid ALL\)]\)
Displays data on page coloring. Table 4–11 shows the command options available with this qualifier.

### Table 4–11 Command Options with the /COLOR and /RAD Qualifiers

<table>
<thead>
<tr>
<th>Options</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/COLOR (^1) with no value</td>
<td>Displays a summary of the lengths of the color(^1) page lists for both free pages and zeroed pages.</td>
</tr>
<tr>
<td>/COLOR(=n) where (n) is a color number</td>
<td>Displays the data in the PFN lists (for the specified color) for both free and zeroed pages.</td>
</tr>
<tr>
<td>/COLOR(=)ALL</td>
<td>Displays the data in the PFN lists (for all colors), for both free and zeroed free pages.</td>
</tr>
<tr>
<td>/COLOR(=n) or /COLOR(=)ALL with /FREE or /ZERO</td>
<td>Displays only the data in the PFN list (for the specified color or all colors), for either free or zeroed free pages as appropriate. The qualifiers /BAD and /MODIFIED are ignored with /COLOR(=n) and /COLOR(=)ALL.</td>
</tr>
<tr>
<td>/COLOR without an option specified together with one or more of /FREE, /ZERO, /BAD, or /MODIFIED</td>
<td>Displays the color summary in addition to the display of the requested list(s).</td>
</tr>
</tbody>
</table>

\(^1\)Whenever COLOR is used in this table, RAD is equally applicable, both in the qualifier name and the meaning.

For more information on page coloring, see OpenVMS System Management Utilities Reference Manual: M–Z.

/FREE
Displays the free page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/MAP
Displays the contents of the PFN memory map. On platforms that support it, the I/O space map is also displayed. The /MAP qualifier cannot be combined with any parameters or other qualifiers.

/MODIFIED
Displays the modified page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/PRIVATE [=address]
Displays private PFN lists. If no address is given, all private PFN lists are displayed; if an address is given, only the PFN list whose head is at the given address is displayed.

/RAD \([=\(n\mid ALL\)]\)
Displays data on the disposition of pages among the Resource Affinity Domains on applicable systems. See Table 4–11 for the command options available with this qualifier.
SHOW PFN_DATA

/SYSTEM
Displays the entire PFN database in order by page frame number, starting at PFN 0000.

/UNTESTED
Displays the state of the untested PFN list that was set up for deferred memory testing.

/ZERO
Displays the contents of the zeroed free page list.

Description

For each page frame number it displays, the SHOW PFN_DATA command lists information used in translating physical page addresses to virtual page addresses. The display has two lines of information. Table 4–12 shows the first line’s fields; Table 4–13 shows the second line’s fields.

Table 4–12 Page Frame Number Information—Line One Fields

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFN</td>
<td>Page frame number.</td>
</tr>
<tr>
<td>DB ADDRESS</td>
<td>Address of PFN structure for this page.</td>
</tr>
<tr>
<td>PT PFN</td>
<td>PFN of the page page table page that maps this page.</td>
</tr>
<tr>
<td>BAK</td>
<td>Place to find information on this page when all links to this PTE are broken: either an index into a process section table or the number of a virtual block in the paging file.</td>
</tr>
<tr>
<td>FLINK</td>
<td>Forward link within PFN database that points to the next physical page (if the page is on one of the lists: FREE, MODIFIED, BAD, or ZEROED); this longword also acts as the count of the number of processes that are sharing this global section.</td>
</tr>
<tr>
<td>BLINK</td>
<td>Backward link within PFN database (if the page is on one of the lists: FREE, MODIFIED, BAD, or ZEROED); also acts as an index into the working set list.</td>
</tr>
<tr>
<td>SWP/BO</td>
<td>Either a swap file page number or a buffer object reference count, depending on a flag set in the page state field.</td>
</tr>
<tr>
<td>LOC</td>
<td>Location of the page within the system. Table 4–10 shows the possible locations with their meaning.</td>
</tr>
<tr>
<td>FLAGS</td>
<td>The flags in text form that are set in page state. Table 4–14 shows the possible flags and their meaning.</td>
</tr>
</tbody>
</table>
Table 4–13  Page Frame Number Information—Line Two Fields

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td></td>
</tr>
<tr>
<td>PTE ADDRESS</td>
<td>Virtual address of the page table entry that describes the virtual page mapped into this physical page. If no virtual page is mapped into this physical page then &quot;&lt;no backpointer&gt;&quot; is displayed.</td>
</tr>
<tr>
<td>PTE Type</td>
<td>If a virtual page is mapped into this physical page, a description of the type of PTE is provided in the next two or three columns: one of &quot;System-space PTE&quot;, &quot;Global PTE&quot;, &quot;Process PTE (process index nnnn)&quot;. If no virtual page is mapped into this physical page, these columns are left blank.</td>
</tr>
<tr>
<td>REFCNT</td>
<td>Number of references being made to this page.</td>
</tr>
<tr>
<td>PAGETYP</td>
<td>Type of physical page. See Table 4-9 for the types of physical pages and their meanings.</td>
</tr>
</tbody>
</table>
| FLAGS          | If the page is a page table page, then the contents of the PRNW_PT_VAL_CNT, PFNW_PT_LCK_CNT, and PFNW_PT_WIN_CNT fields are displayed. The format is as follows:  

\[
\text{VALCNT} = nnnn \quad \text{LCKCNT} = nnnn \quad \text{WINCNT} = nnnn
\]

---

Table 4–14  Flags Set in Page State

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFOBJ</td>
<td>Set if any buffer objects reference this page</td>
</tr>
<tr>
<td>COLLISION</td>
<td>Indicates an empty collision queue when page read is complete</td>
</tr>
<tr>
<td>BADPAG</td>
<td>Indicates a bad page</td>
</tr>
<tr>
<td>RPTEVT</td>
<td>Indicates a report event on I/O completion</td>
</tr>
<tr>
<td>DELCON</td>
<td>Indicates a delete PFN when REFCNT=0</td>
</tr>
<tr>
<td>MODIFY</td>
<td>Indicates a dirty page (modified)</td>
</tr>
<tr>
<td>UNAVAILABLE</td>
<td>Indicates PFN is unavailable; most likely a console page</td>
</tr>
</tbody>
</table>
Examples

1. SDA> SHOW PFN_DATA/MAP

   System Memory Map
   -----------------
   Start PFN PFN count Flags
   --------- --------- -----
   00000000 000000FA 0009 Console Base
   000000FA 00003306 000A OpenVMS Base
   00003C00 000003FF 000A OpenVMS Base
   00003FFF 00000001 0009 Console Base
   00003400 00000800 0010 Galaxy_Shared

   This example shows the output when you invoke the SHOW PFN/MAP command.

2. SDA> SHOW PFN 1B5:1C1

  PFN data base
   --------------
   PFN      DB  ADDRESS     PT PFN        BAK          FLINK    BLINK   SWP/BO   LOC              FLAGS
   -------- ----------------- -------- ----------------- -------- -------- ------ ------- -------------------------------
   000001B5 FFFFFFFE.00004448 000001AB 00000000.00000000 00000000 00000000  ----  ACTIVE
   FFFFFFFD.FFFFFF78 System-space PTE                              0002  SPT(L3) VALCNT=0002 LCRCNT=FFF WINCNT=FFFF
   000001B6 FFFFFFFE.00004470 000001AB 00000000.00000000 00000000 00000000  ----  ACTIVE
   FFFFFFFD.FFFFFF70 System-space PTE                              0002  SPT(L3) VALCNT=0007 LCRCNT=FFF WINCNT=FFFF
   000001B7 FFFFFFFE.00004498 000001AB 00000000.00000000 00000000 00000000  ----  ACTIVE
   FFFFFFFD.FFFFFF78 System-space PTE                              0002  SPT(L3) VALCNT=FPP WINCNT=FFFF
   000001B8 FFFFFFFE.000044C0 00000EAC 000000ED.00000000 00000001 00000000  ----  ACTIVE
   FFFFFFFE.00056430 Global PTE                                    0001  GLOBAL
   000001B9 FFFFFFFE.000044E8 00000000 00000000.00000000 00000000 00000000  ----  ACTIVE
   <no backpointer>                                               0001  SYSTEM
   000001BA FFFFFFFE.00004510 00001662 03000000.00000000 00001134 000016EC  ----  MFLIST modify
   FFFFFFFD.FFFFFE00 Process PTE (process index 001B)              0000  PROCESS
   000001Bb FFFFFFFE.00004538 00000D87 00000000.00000000 0000126A 00000C78  ----  FRELIST
   FFFFFFFE.00005F18 Global PTE                                    0000  GLOBAL
   000001Bc FFFFFFFE.00004560 00001738 03000000.00000000 00000001 00000000  ----  ACTIVE modify
   FFFFFFFE.00008BB0 Global PTE                                   0001  GLOBAL
   000001Bd FFFFFFFE.00004588 00000185 03000000.00000000 00000000 00000000  ----  ACTIVE
   FFFFFFFE.0000008A Process PTE (process index 001F)              0001  PROCESS
   000001Be FFFFFFFE.000045B0 000017A8 03000000.00000000 00000000 00000000  ----  ACTIVE
   FFFFFFFE.000056DE Process PTE (process index 001B)              0000  PROCESS
   000001Bf FFFFFFFE.000045E0 00001738 03000000.00000000 00000000 00000000  ----  ACTIVE
   FFFFFFFD.FFFFFE00 System-space PTE                              0001  SPT(L3) VALCNT=FFF LCRCNT=FPP WINCNT=FFFF
   000001C0 FFFFFFFE.00004600 000012A2 0000017.00000000 000012D8 000011E8  ----  FRELIST
   FFFFFFFE.000012E0 Process PTE (process index 0004)              0000  PROCESS
   000001C1 FFFFFFFE.00004628 000017A8 00000168.00000000 00000002 00000000  ----  ACTIVE
   FFFFFFFE.0005BAC8 Global PTE                                   0001  GLOBAL

   This example shows the output from SHOW PFN for a range of pages.
SHOW POOL
Displays the contents of the nonpaged dynamic storage pool, the bus-addressable pool, and the paged dynamic storage pool. You can display part or all of each pool. If you do not specify a range or qualifiers, the default is SHOW POOL/ALL. Optionally, you can display the pool history ring buffer and pool statistics.

Format

Parameter
range
Range of virtual addresses in pool that SDA is to examine. You can express a range using the following syntax:
m:n Range of virtual addresses in pool from m to n
m;n Range of virtual addresses in pool starting at m and continuing for n bytes

Qualifiers
/ALL
Displays the entire contents of dynamic storage pool, except for those portions that are free (available). This is the default behavior of the SHOW POOL command.

/BAP
Displays the contents of the bus-addressable dynamic storage pool currently in use.

/BRIEF
Displays only general information about dynamic storage pool and its addresses.

/CHECK
Checks all free packets for POOLCHECK-style corruption, in exactly the same way that the system does when generating a POOLCHECK crashdump.

/FREE
Displays the entire contents, both allocated and free, of the specified region or regions of pool. Use the /FREE qualifier with a range to show all of the used and free pool in the given range.

/HEADER
Displays only the first 16 longwords of each data packet found within the specified region or regions of pool.

/MAXIMUM_BYTES [=n]
Displays only the first n bytes of a pool packet; default is 64 bytes.

/NONPAGED
Displays the contents of the nonpaged dynamic storage pool currently in use.
SDA Commands
SHOW POOL

/PAGED
Displays the contents of the paged dynamic storage pool currently in use.

/RING_BUFFER
Displays the contents of the nonpaged pool history ring buffer if pool checking has been enabled. Entries are displayed in reverse chronological order, that is, most to least recent.

/STATISTICS [= ALL]
Displays usage statistics about each lookaside list and the variable free list. For each lookaside list, its queue header address, packet size, the number of packets, attempts, fails, and deallocations are displayed. (If pool checking is disabled, the attempts, fails, and deallocations are not displayed.) For the variable free list, its queue header address, the number of packets and the size of the smallest and largest packets are displayed. /STATISTICS can be further qualified by using either /NONPAGED, /BAP, or /PAGED to display statistics for a specified pool area. (Note that paged pool has no lookaside lists; therefore, only variable free list statistics are displayed.)

If /STATISTICS is specified without the ALL keyword, only active lookaside lists are displayed. Use /STATISTICS = ALL to display all lookaside lists.

/SUBTYPE=packet-type
Displays the packets within the specified region or regions of pool that are of the indicated packet-type. For information on packet-type, see packet-type in the Description section.

/SUMMARY
Displays only an allocation summary for each specified region of pool.

/TYTP=packet-type
Displays the packets within the specified region or regions of pool that are of the indicated packet-type. For information on packet-type, see packet-type in the Description section.

/UNUSED
Displays only variable free packets and lookaside list packets, not used packets.

Description
The SHOW POOL command displays information about the contents of any specified region of dynamic storage pool. There are several distinct display formats, as follows:

- Pool layout display. This display includes the addresses of the pool structures and lookaside lists, and the ranges of memory used for pool.
- Full pool packet display. This display has a section for each packet, consisting of a summary line (the packet type, its start address and size, and, on systems that have multiple Resource Affinity Domains (RADs), the RAD number), followed by a dump of the contents of the packet in hexadecimal and ASCII.
- Header pool packet display. This display has a single line for each packet. This line contains the packet type, its start address and size, and, on systems that have multiple RADs, the RAD number, followed by the first sixteen bytes of the packet, in hexadecimal and ASCII.
• Pool summary display. This display consists of a single line for each packet type, and includes the type, the number of occurrences and the total size, and the percentage of used pool consumed by this packet type.

• Pool statistics display. This display consists of statistics for variable free pool and for each lookaside list. For variable free pool, it includes the number of packets, the total bytes available, and the sizes of the smallest and largest packets. In addition, if pool checking is enabled, the total bytes allocated from the variable list and the number of times pool has been expanded are also displayed.

For lookaside lists, the display includes the listhead address and size, the number of packets (both the maintained count and the actual count), the operation sequence number for the list, the allocation attempts and failures, and the number of deallocations.

On systems with multiple RADs, statistics for on-RAD deallocations are included in the display for the first RAD.

• Ring buffer display. This display is only available when pool checking is enabled. It consists of one line for each packet in the ring buffer and includes the address and size of the pool packet being allocated or deallocated, its type, the PC of the caller and the pool routine called, the CPU and IPL of the call, and the system time.

The qualifiers used on the SHOW POOL command determine which displays are generated. The default is the pool layout display, followed by the full pool packet display, followed by the pool summary display, these being generated in turn for Nonpaged Pool, Bus-Addressable Pool (if it exists in the system or dump being analyzed), and then Paged Pool.

If a range, type, or subtype is specified, then the pool layout display is not generated, and the pool summary display is a summary only for the range and/or type/subtype, and not for the entire pool.

Note that not all displays are relevant for all pool types. For example, Paged Pool has no lookaside lists, so the Paged Pool statistics display consists only of variable free pool information. And since there is a single ring buffer for all pools, only one ring buffer display is generated even if all pools are being displayed.

Packet-type
Each packet of pool has a type field (a byte containing a value in the range of 0-255). Many of these type values have names associated that are defined in $DYNDEF in SYS$LIBRARY:LIB.MLB. The packet-type specified in the /TYPE qualifier of the SHOW POOL command can either be the value of the pool type or its associated name.

Some pool packet-types have an additional subtype field (also a byte containing a value in the range of 0-255), many of which also have associated names. The packet-type specified in the /SUBTYPE qualifier of the SHOW POOL command can either be the value of the pool type or its associated name. However, if given as a value, a /TYPE qualifier (giving a value or name) must also be specified. Note also that /TYPE and /SUBTYPE are interchangeable if the packet-type is given by name. Table 4–15 shows several examples.
### Table 4–15 /TYPE and /SUBTYPE Qualifier Examples

<table>
<thead>
<tr>
<th>/TYPE and /SUBTYPE Qualifiers</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TYPE = CI</td>
<td>All CI packets regardless of subtype</td>
</tr>
<tr>
<td>/TYPE = CI_MSG</td>
<td>All CI packets with subtype CI_MSG</td>
</tr>
<tr>
<td>/TYPE = MISC/SUBTYPE = 120</td>
<td>All MISC packets with subtype 120</td>
</tr>
<tr>
<td>/TYPE = 0 or /TYPE = UNKNOWN</td>
<td>All packets with an unknown TYPE/SUBTYPE combination</td>
</tr>
</tbody>
</table>

**SDA Commands**

**SHOW POOL**

---

4–142  SDA Commands
Examples

1. SDA> SHOW POOL

Non-Paged Dynamic Storage Pool

<table>
<thead>
<tr>
<th>NPPOOL address:</th>
<th>Pool map address:</th>
<th>Number of lookaside lists:</th>
<th>Granularity size:</th>
<th>Ring buffer address:</th>
<th>Most recent ring buffer entry:</th>
</tr>
</thead>
<tbody>
<tr>
<td>81009088</td>
<td>81562900</td>
<td>128</td>
<td>64</td>
<td>81552200</td>
<td>815553A0</td>
</tr>
</tbody>
</table>

LSTHDS(s)

<table>
<thead>
<tr>
<th>RAD</th>
<th>LSTHDS</th>
<th>Variable</th>
<th>Lookaside</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>00</td>
<td>FFFFFFF.81008830</td>
<td>FFFFFFF.8100883C</td>
<td>FFFFFFF.81008868</td>
</tr>
<tr>
<td>01</td>
<td>FFFFFFF.7FFFFE000</td>
<td>FFFFFFF.7FFFFE00C</td>
<td>FFFFFFF.7FFFFE038</td>
</tr>
<tr>
<td>02</td>
<td>FFFFFFF.7FFFC000</td>
<td>FFFFFFF.7FFFC00C</td>
<td>FFFFFFF.7FFFC038</td>
</tr>
<tr>
<td>03</td>
<td>FFFFFFF.7FFFA000</td>
<td>FFFFFFF.7FFFA00C</td>
<td>FFFFFFF.7FFFA038</td>
</tr>
</tbody>
</table>

Segment(s)

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Length</th>
<th>RAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>81548000</td>
<td>8172B9FF</td>
<td>001E3A00</td>
<td>00</td>
</tr>
<tr>
<td>81735A00</td>
<td>8173D53F</td>
<td>00007B40</td>
<td>00</td>
</tr>
<tr>
<td>81755DC0</td>
<td>81AFDFFF</td>
<td>003A8240</td>
<td>00</td>
</tr>
<tr>
<td>81AFE000</td>
<td>81C43FFF</td>
<td>00146000</td>
<td>01</td>
</tr>
<tr>
<td>81D8A000</td>
<td>81ECFFFF</td>
<td>00146000</td>
<td>03</td>
</tr>
<tr>
<td>81ED0000</td>
<td>81F1FFFF</td>
<td>00050000</td>
<td>02</td>
</tr>
</tbody>
</table>

Per-RAD Totals

<table>
<thead>
<tr>
<th>RAD</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00598000</td>
</tr>
<tr>
<td>01</td>
<td>00146000</td>
</tr>
<tr>
<td>02</td>
<td>00146000</td>
</tr>
<tr>
<td>03</td>
<td>00146000</td>
</tr>
</tbody>
</table>

Non-Paged total: 0098A000

Dump of packets allocated from Non-Paged Pool

<table>
<thead>
<tr>
<th>Packet: MP_CPU</th>
<th>Start address: 81548000</th>
<th>Length: 000009C0</th>
<th>RAD: 00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00000000 00000000 00000003E 00000001 00000002 026A09C0 ACD1A180 81C52F40</td>
<td>8/A...j@...j.---------g--------</td>
<td>81548000</td>
</tr>
<tr>
<td></td>
<td>81548038 81548038 81548030 81548030 81548028 81548028 81548028 81548028 81548028 81548028 81548028 81548028 81548028 81548028 81548028</td>
<td></td>
<td>81548020</td>
</tr>
<tr>
<td></td>
<td>81548058 81548058 81548050 81548050 81548048 81548048 81548048 81548048 81548048 81548048 81548048 81548048 81548048 81548048 81548048</td>
<td></td>
<td>81548040</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet: Unknown</th>
<th>Start address: 815489C0</th>
<th>Length: 00000180</th>
<th>RAD: 00</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFF AD332000 00500000 00000820 PFFFFFFFF 81548800 PFFFFFFFF 81548800</td>
<td>..T.........T.....P.. 3....</td>
<td>815489C0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet: DDB</th>
<th>Start address: 81548B40</th>
<th>Length: 00000300</th>
<th>RAD: 00</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD410000 81564480 81548BC0 008F4240 00000000 63D60300 088B798F 962DA431</td>
<td>1e--y........c... @8..A.T.1D...A-</td>
<td>81548B40</td>
<td></td>
</tr>
</tbody>
</table>
Summary of Non-Paged Pool contents
----------------------------------
Packet type/subtype   Packet count    Packet bytes    Percent
---------------------------  ----------------  ----------------  --------
Unknown                  0000001E4          00145BC0           (50.7%)
ADP                     00000009           00000A00            (0.1%)
ACB                     0000008D           00002500            (0.4%)
AQR                     00000002           00001080            (0.2%)
LOADCODE                0000003D           00001440            (0.7%)
LDRIMG                  00000003D          000004C40           (0.7%)
INIT                    00000008           00003880            (0.6%)
PCBVEC                 000000001           00001840            (0.2%)
PHVEC                  000000001           00000700            (0.1%)
MPWMAP                 000000005           000006C0            (0.2%)
PRCMAP                 000000001           00000080            (0.0%)

Total space used: 002825C0 (2631104.) bytes out of 009BA000 (10199040.) bytes
in 0000184C (6220.) packets
Total space utilization: 25.8%

This example shows the Nonpaged Pool portion of the default SHOW POOL
display.

2. SDA> SHOW POOL/TYPE=IPC/HEADER 8156E1A0:815912C0

Non-Paged Dynamic Storage Pool
-------------------------------
Dump of packets allocated from Non-Paged Pool
---------------------------------------------
Packet type/subtype   Start    Length   RAD   Header contents
---------------------- -------- -------- ---- ---------------------------------------------
IPC_TDB               8156E140 00000040 00 81591180 057B0040 00000040 81591180 ..Y.@...@.{...Y.
IPC_LIST              815838C0 00009840 00 004C0200 087B9840 0057A740 8158D100 .NX.@@W.B......
IPC_LIST              8158D100 00001840 00 00040400 087B1840 00570F00 8158E940 @éX...W.Ø......
IPC_LIST              8158E940 00002840 00 00140200 087B2840 0056F6C0 81591180 ..Y.AB.V.Ø......
IPC_TPCB              81591180 00000080 00 00000000 067B0080 0056CE80 81591200 ..Y..IV......
IPC                  81591200 000000C0 00 00000000 007B00C0 0056CE00 815912C0 A.Y..IV.A......

Summary of Non-Paged Pool contents
----------------------------------
Packet type/subtype   Packet count    Packet bytes    Percent
---------------------------  ----------------  --------
IPC                     00000006           0000DA40  (100.0%)  
IPC                     00000001           00000010  (0.3%)  
IPC_TDB                 00000001           00000040  (0.1%)  
IPC_TPCB                00000001           00000080  (0.2%)  
IPC_LIST                00000003           000008C0  (99.3%)  

Total space used: 0000DA40 (55872.) bytes out of 00023180 (143744.) bytes
in 00000006 (6.) packets
Total space utilization: 38.9%

This example shows how a pool packet type and a range of addresses can be
specified.
3. SDA> SHOW POOL/STATISTICS

Non-Paged Pool statistics for RAD 00
------------------------------------

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-RAD deallocations (all RADs):</td>
<td>1221036</td>
</tr>
<tr>
<td>Total deallocations (all RADs):</td>
<td>1347991</td>
</tr>
<tr>
<td>Percentage of on-RAD deallocations:</td>
<td>90.6%</td>
</tr>
</tbody>
</table>

Variable list statistics
------------------------

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of packets on variable list:</td>
<td>7</td>
</tr>
<tr>
<td>Total bytes on variable list:</td>
<td>3613376</td>
</tr>
<tr>
<td>Smallest packet on variable list:</td>
<td>256</td>
</tr>
<tr>
<td>Largest packet on variable list:</td>
<td>3598016</td>
</tr>
<tr>
<td>Bytes allocated from variable list:</td>
<td>2140480</td>
</tr>
<tr>
<td>Times pool expanded:</td>
<td>0</td>
</tr>
</tbody>
</table>

Lookaside list statistics
-------------------------

<table>
<thead>
<tr>
<th>Listhead address</th>
<th>List size</th>
<th>Packets (approx)</th>
<th>Packets (actual)</th>
<th>Operation sequence #</th>
<th>Allocation attempts</th>
<th>Allocation failures</th>
<th>Deallocs</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFF.81C65F40</td>
<td>64</td>
<td>5</td>
<td>5</td>
<td>10005</td>
<td>10549</td>
<td>492</td>
<td>10062</td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>128</td>
<td>21</td>
<td>21</td>
<td>366</td>
<td>4881</td>
<td>4515</td>
<td>387</td>
</tr>
<tr>
<td>FFFFFFFF.81C45A40</td>
<td>192</td>
<td>33</td>
<td>33</td>
<td>27376</td>
<td>27542</td>
<td>166</td>
<td>27409</td>
</tr>
<tr>
<td>FFFFFFFF.81C65F40</td>
<td>320</td>
<td>SECURITY_PSB</td>
<td>80283A9C NSA_STD$FREE_PSB_C+0024C DEALLO_POOL_NPP</td>
<td>0   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>192</td>
<td>SECURITY_PXB_ARRAY</td>
<td>80283A30 NSA_STD$FREE_PSB_C+001E0 DEALLO_POOL_NPP</td>
<td>0   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.81C45A40</td>
<td>64</td>
<td>ACB</td>
<td>8014A09C SCH$INIT_C+00F18 DEALLO_POOL_NPP_SIZ</td>
<td>2   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>140</td>
<td>SECURITY_PXB_ARRAY</td>
<td>80283B8C NSA$GET_PSB_C+0005C ALLO_POOL_NPP</td>
<td>0   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.81C65F40</td>
<td>320</td>
<td>SECURITY_PSB</td>
<td>80283B70 NSA$GET_PSB_C+00040 ALLO_POOL_NPP_SIZ</td>
<td>2   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>192</td>
<td>SECURITY_PXB_ARRAY</td>
<td>80283B70 NSA$GET_PSB_C+00040 ALLO_POOL_NPP_SIZ</td>
<td>2   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFFFFFFF.81C45A40</td>
<td>64</td>
<td>ACB</td>
<td>801281F8 PROCESS_MANAGEMENT_MON+00F18 ALLO_POOL_NPP_SIZ</td>
<td>2   8  009F1E47.549449F0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This example shows the Nonpaged Pool portion of the SHOW POOL/STATISTICS display.

4. SDA> SHOW POOL/RING_BUFFER

Pool History Ring-Buffer
------------------------

<table>
<thead>
<tr>
<th>Packet</th>
<th>Size</th>
<th>Type/Subtype</th>
<th>Caller's PC</th>
<th>Operation</th>
<th>IPL</th>
<th>CPU</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFF.81C65F40</td>
<td>320</td>
<td>SECURITY_PSB</td>
<td>80283A9C NSA_STD$FREE_PSB_C+0024C</td>
<td>DEALLO_POOL_NPP</td>
<td>0</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>192</td>
<td>SECURITY_PXB_ARRAY</td>
<td>80283A30 NSA_STD$FREE_PSB_C+001E0</td>
<td>DEALLO_POOL_NPP</td>
<td>0</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
<tr>
<td>FFFFFFFF.81C45A40</td>
<td>64</td>
<td>ACB</td>
<td>8014A09C SCH$INIT_C+00F18</td>
<td>DEALLO_POOL_NPP_SIZ</td>
<td>2</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>140</td>
<td>SECURITY_PXB_ARRAY</td>
<td>80283B8C NSA$GET_PSB_C+0005C</td>
<td>ALLO_POOL_NPP</td>
<td>0</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
<tr>
<td>FFFFFFFF.81C65F40</td>
<td>320</td>
<td>SECURITY_PSB</td>
<td>80283B70 NSA$GET_PSB_C+00040</td>
<td>ALLO_POOL_NPP_SIZ</td>
<td>2</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
<tr>
<td>FFFFFFFF.81C44E00</td>
<td>192</td>
<td>SECURITY_PXB_ARRAY</td>
<td>80283B70 NSA$GET_PSB_C+00040</td>
<td>ALLO_POOL_NPP_SIZ</td>
<td>2</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
<tr>
<td>FFFFFFFF.81C45A40</td>
<td>64</td>
<td>ACB</td>
<td>801281F8 PROCESS_MANAGEMENT_MON+00F18</td>
<td>ALLO_POOL_NPP_SIZ</td>
<td>2</td>
<td>8</td>
<td>009F1E47.549449F0</td>
</tr>
</tbody>
</table>

This example shows the output of the SHOW POOL/RING_BUFFER display.
SHOW PORTS

Displays those portions of the port descriptor table (PDT) that are port independent.

Format

SHOW PORTS  [/qualifier[,...]]

Parameters

None.

Qualifiers

/ADDRESS=pdt-address
Displays the specified port descriptor table (PDT). You can find the pdt-address for any active connection on the system in the PDT summary page display of the SHOW PORTS command. This command also defines the symbol PE_PDT. The connection descriptor table (CDT) addresses are also stored in many individual data structures related to System Communications Services (SCS) connections, for instance, in the path block displays of the SHOW CLUSTER/SCS command.

/BUS=bus-address
Displays bus (LAN device) structure data.

/CHANNEL=channel-address
Displays channel (CH) data.

/DEVICE
Displays the network path description for a channel.

/MESSAGE
Displays the message data associated with a virtual circuit (VC).

/NODE=node
Shows only the virtual circuit block associated with the specific node. When you use the /NODE qualifier, you must also specify the address of the PDT using the /ADDRESS qualifier.

/VC=vc-address
Displays the virtual circuit data.

Description

The SHOW PORTS command provides port-independent information from the port descriptor table (PDT) for those CI ports with full System Communications Services (SCS) connections. This information is used by all SCS port drivers.

Note that the SHOW PORTS command does not display similar information about UDA ports, BDA ports, and similar controllers.
The SHOW PORTS command also defines symbols for PEDRIVER based on the cluster configuration. These symbols include the following information:

- Virtual circuit (VC) control blocks for each of the remote systems
- Bus data structure for each of the local LAN adapters
- Some of the data structures used by both PEDRIVER and the LAN drivers

The following symbols are defined automatically:

- VC_nodename—Example: VC_NODE1, address of the local node's virtual circuit to node NODE1.
- CH_nodename—The preferred channel for the virtual circuit. For example, CH_NODE1, address of the local node's preferred channel to node NODE1.
- BUS_busname—Example: BUS_ETA, address of the local node's bus structure associated with LAN adapter ETA0.
- PE_PDT—Address of PEDRIVER's port descriptor table.
- MGMT_VCRP_busname—Example: MGMT_VCRP_ETA, address of the management VCRP for bus ETA.
- HELLO_VCRP_busname—Example: HELLO_VCRP_ETA, address of the HELLO message VCRP for bus ETA.
- VCIIB_busname—Example: VCIIB_ETA, address of the VCIB for bus ETA.
- UCB_LAVC_busname—Example: UCB_LAVC_ETA, address of the LAN device's UCB used for the local-area OpenVMS Cluster protocol.
- UCB0_LAVC_busname—Example: UCB0_LAVC_ETA, address of the LAN device's template UCB.
- LDC_LAVC_busname—Example: LDC_LAVC_ETA, address of the LDC structure associated with LAN device ETA.
- LSB_LAVC_busname—Example: LSB_LAVC_ETA, address of the LSB structure associated with LAN device ETA.

These symbols equate to system addresses for the corresponding data structures. You can use these symbols, or an address, in SHOW PORTS qualifiers that require an address, as for example:

```
SDA > SHOW PORTS/ADDRESS=BUS=BUS_ETA
```

The SHOW PORTS command produces several displays. The initial display, the **PDT summary page**, lists the PDT address, port type, device name, and driver name for each PDT. Subsequent displays provide information taken from each PDT listed on the summary page.

You can use the **ADDRESS** qualifier to the SHOW PORTS command to produce more detailed information about a specific port. The first display of the SHOW PORTS/ADDRESS command duplicates the last display of the SHOW PORTS command, listing information stored in the port's PDT. Subsequent displays list information about the port blocks and virtual circuits associated with the port.
SDA Commands
SHOW PORTS

Examples

1. SDA > SHOW PORTS

VMScluster data structures
--------------------------

--- PDT Summary Page ---

<table>
<thead>
<tr>
<th>PDT Address</th>
<th>Type</th>
<th>Device</th>
<th>Driver Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>80E2A180</td>
<td>pn</td>
<td>PNA0</td>
<td>SYSSPNDRIVER</td>
</tr>
<tr>
<td>80EC3C70</td>
<td>pe</td>
<td>PEA0</td>
<td>SYSSPEDRIVER</td>
</tr>
</tbody>
</table>

--- Port Descriptor Table (PDT) 80E2A180 ---

Type: 09 pn
Characteristics: 0000

Msg Header Size 104 Flags 0000 Message Sends 3648575
Max Xfer Bcnt 00100000 Counter CDRP 00000000 Message Recvs 4026887
Poller Sweep 21 Load Vector 80E2DFCC Mess Sends NoFP 3020422
Fork Block W.Q. 80E2A270 Load Class 60 Mess Recvs NoFP 3398732
UCB Address 80E23380 Connection W.Q. 80E4BF94 Datagram Sends 0
ADP Address 80E2A2E0 Yellow Q. 80E2A2E0 Datagram Recvs 0
Max VC timeout 16 Red Q. 80E2A2E8 Portlock 80E1ED80
SCS Version 2 Disabled Q. 80FABB74 Res Bundle Size 208
UCB Address 80E2A2E0 Connection W.Q. 80E4BF94 Datagram Sends 0
ADP Address 80E2A2E0 Yellow Q. 80E2A2E0 Datagram Recvs 0
Max VC timeout 16 Red Q. 80E2A2E8 Portlock 80E1ED80
SCS Version 2 Disabled Q. 80FABB74 Res Bundle Size 208

--- Port Descriptor Table (PDT) 80EC3C70 ---

Type: 03 pe
Characteristics: 0000

Msg Header Size 32 Flags 0000 Message Sends 863497
Max Xfer Bcnt FFFFFFFF Counter CDRP 00000000 Message Recvs 886284
Poller Sweep 30 Load Vector 80E5DF6C Mess Sends NoFP 863497
Fork Block W.Q. 80EC3D60 Load Class 10 Mess Recvs NoFP 886284
UCB Address 80EC33C0 Connection W.Q. 80E5DFD4 Datagram Sends 0
ADP Address 80EC33C0 Yellow Q. 80EC3D0 Datagram Recvs 0
Max VC timeout 16 Red Q. 80EC3D80 Portlock 00000000
SCS Version 2 Disabled Q. 812E72B4 Res Bundle Size 0
Port Map 00000000

This example illustrates the default output of the SHOW PORTS command.
### VMScluster data structures

--- Port Descriptor Table (PDT) 80EC3C70 ---

<table>
<thead>
<tr>
<th>Type:</th>
<th>03 pe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristics:</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Msg Header Size</th>
<th>32</th>
<th>Flags</th>
<th>0000</th>
<th>Message Sends</th>
<th>864796</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Xfer Bcnt</td>
<td>FF</td>
<td>Counter CDRP</td>
<td>00000000</td>
<td>Message Recvs</td>
<td>887086</td>
</tr>
<tr>
<td>Poller Sweep</td>
<td>30</td>
<td>Load Vector</td>
<td>80EDBFB8</td>
<td>Mess Sends NoFP</td>
<td>864796</td>
</tr>
<tr>
<td>Fork Block W.Q.</td>
<td>80EC3D60</td>
<td>Load Class</td>
<td>10</td>
<td>Mess Recvs NoFP</td>
<td>887086</td>
</tr>
<tr>
<td>UCB Address</td>
<td>80EC33C0</td>
<td>Connection W.Q.</td>
<td>80EFF5D4</td>
<td>Datagram Sends</td>
<td>0</td>
</tr>
<tr>
<td>ADP Address</td>
<td>00000000</td>
<td>Yellow Q.</td>
<td>80EC3DD0</td>
<td>Datagram Recvs</td>
<td>0</td>
</tr>
<tr>
<td>Max VC timeout</td>
<td>16</td>
<td>Red Q.</td>
<td>80EC3DD8</td>
<td>Portlock</td>
<td>00000000</td>
</tr>
<tr>
<td>SCS Version</td>
<td>2</td>
<td>Disabled Q.</td>
<td>812E72B4</td>
<td>Res Bundle Size</td>
<td>0</td>
</tr>
</tbody>
</table>

--- Port Block 80EC4540 ---

<table>
<thead>
<tr>
<th>Status:</th>
<th>0001 authorize</th>
</tr>
</thead>
<tbody>
<tr>
<td>VC Count:</td>
<td>20</td>
</tr>
<tr>
<td>Secs Since Last Zeroed:</td>
<td>77020</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SBUF Size</th>
<th>824</th>
<th>LBUF Size</th>
<th>5042</th>
<th>Fork Count</th>
<th>1943885</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBUF Count</td>
<td>28</td>
<td>LB Buf Count</td>
<td>1</td>
<td>Refork Count</td>
<td>0</td>
</tr>
<tr>
<td>SBUF Max</td>
<td>768</td>
<td>LB Buf Max</td>
<td>384</td>
<td>Last Refork</td>
<td>00000000</td>
</tr>
<tr>
<td>SBUF Quo</td>
<td>28</td>
<td>SB Buf Quo</td>
<td>1</td>
<td>SCS Messages</td>
<td>1154378</td>
</tr>
<tr>
<td>SBUF Miss</td>
<td>1871</td>
<td>LB Buf Miss</td>
<td>3408</td>
<td>VC Queue Cnt</td>
<td>361349</td>
</tr>
<tr>
<td>SBUF Allocs</td>
<td>1676801</td>
<td>LB Buf Allocs</td>
<td>28596</td>
<td>TQE Received</td>
<td>770201</td>
</tr>
<tr>
<td>SBUF's In Use</td>
<td>2</td>
<td>LB Buf's In Use</td>
<td>0</td>
<td>Timer Done</td>
<td>770201</td>
</tr>
<tr>
<td>Peak SBUF In Use</td>
<td>101</td>
<td>Peak LB Buf In Use</td>
<td>10</td>
<td>RWAITQ Count</td>
<td>30288</td>
</tr>
<tr>
<td>SBUF Queue Empty</td>
<td>0</td>
<td>LB Buf Queue Empty</td>
<td>0</td>
<td>LDL Buf/Msg</td>
<td>32868</td>
</tr>
<tr>
<td>TR SBUF Queue Empty</td>
<td>0</td>
<td>Ticks/Second</td>
<td>10</td>
<td>ACK Delay</td>
<td>1000000</td>
</tr>
<tr>
<td>No SBUF for ACK</td>
<td>0</td>
<td>Listen Timeout</td>
<td>8</td>
<td>Hello Interval</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bus Addr</th>
<th>Bus</th>
<th>LAN Address</th>
<th>Error Count</th>
<th>Last Error</th>
<th>Time of Last Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>80EC4C00</td>
<td>LCL</td>
<td>00-00-00-00-00-00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80EC5400</td>
<td>EXA</td>
<td>08-00-2B-17-CF-92</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80EC5F40</td>
<td>FXA</td>
<td>08-00-2B-29-E1-40</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- Virtual Circuit (VC) Summary ---

<table>
<thead>
<tr>
<th>VC Addr</th>
<th>Node</th>
<th>SCS ID</th>
<th>Lcl ID</th>
<th>Status Summary</th>
<th>Last Event Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>80E566C0</td>
<td>ARUSHA</td>
<td>19617</td>
<td>223/DF</td>
<td>open/path</td>
<td>8-FEB-2001 16:01:57.58</td>
</tr>
<tr>
<td>80E98840</td>
<td>ETOSHA</td>
<td>19699</td>
<td>222/DE</td>
<td>open/path</td>
<td>8-FEB-2001 16:01:58.41</td>
</tr>
<tr>
<td>80E98A80</td>
<td>VMS</td>
<td>19578</td>
<td>221/DD</td>
<td>open/path</td>
<td>8-FEB-2001 16:01:58.11</td>
</tr>
</tbody>
</table>

This example illustrates the output produced by the SHOW PORTS command for the PDT at address 80EC3C70.
SHOW PROCESS

Displays the software and hardware context of any process in the balance set.

Format

SHOW PROCESS { [process-name | ALL] 
| /ADDRESS=pcb_address | /ID=nn | /INDEX=nn | /SYSTEM} 
| /ALL | /BUFFER_OBJECTS | /CHANNEL 
| /FANDLES | /FID_ONLY | /GSTX=index | /IMAGES [=ALL] 
| /INVALID_PFN [=option] | /NEXT 
| /NONMEMORY_PFN [=option] 
| /LOCKS | /BRIEF | /L1 | /L2 | /L3 
| /PAGE_TABLES | /P0 | /P1 | /P2 | /PT | /PCB 
| /PERSONA [=address][/RIGHTS [=AUTHORIZED]] 
| /PHD | /PROCESS_SECTION_TABLE | /PST 
| /PTE_ADDRESS | /RDE [=id] 
| /REGIONS [=id] 
| /REGISTERS | /RMS [=option[...]] | /SECTION_INDEX=n 
| /SEMAPHORE | /THREADS 
| /WORKING_SET_LIST]

Parameters

ALL
Information about all processes that exist in the system.

process-name
Name of the process for which information is to be displayed. Use of the process-name parameter, the ADDRESS qualifier, the INDEX qualifier, or the SYSTEM qualifier causes the SHOW PROCESS command to perform an implicit SET PROCESS command, making the indicated process the current process for subsequent SDA commands. You can determine the names of the processes in the system by issuing a SHOW SUMMARY command.

The process-name can contain up to 15 letters and numerals, including the underscore (_) and dollar sign ($). If it contains any other characters, you must enclose the process-name in quotation marks (" ").

Qualifiers

/ADDRESS=pcb-address
Specifies the process control block (PCB) address of a process in order to display information about the process.

/ALL
Displays all information shown by the following qualifiers:

/CHANNEL 
/BUFFER_OBJECTS 
/FANDLES 
/IMAGES [=ALL] 
/LOCKS 
/PAGE_TABLES 
/PCB 
/PERSONA/RIGHTS
SHOW PROCESS

/PHD
/PROCESS_SECTION_TABLE
/REGIONS
/REGISTERS
/RMS
/SEMAPHORE
/THREADS
/WORKING_SET_LIST

/AUTHORIZED
Used with the /PERSONA/RIGHTS qualifiers. See the /PERSONA/RIGHTS/AUTHORIZED description for the use of the /AUTHORIZED qualifier.

/BRIEF
When used with the /LOCKS qualifier, causes SDA to display each lock owned by the current process in brief format, that is, one line for each lock.

/BUFFER_OBJECTS
Displays all the buffer objects that a process has created.

/CHANNEL
Displays information about the I/O channels assigned to the process.

/FANDLES
Displays the data on the process's fast I/O handles.

/FID_ONLY
When used with /CHANNEL or /PROCESS_SECTION_TABLE (/PST), causes SDA to not attempt to translate the FID (File ID) to a file name when invoked with ANALYZE/SYSTEM.

/GSTX=index
When used with the /PAGE_TABLES qualifier, causes SDA to display only page table entries for the specific global section.

/IMAGES [= ALL]
For all images in use by this process, displays the address of the image control block, the start and end addresses of the image, the activation code, the protected and shareable flags, the image name, and the major and minor IDs of the image. The /IMAGES = ALL qualifier also displays the base, end, image offset, and section type for installed resident images in use by this process.

See the OpenVMS Linker Utility Manual and the Install utility chapter in the OpenVMS System Management Utilities Reference Manual for more information on images installed using the /RESIDENT qualifier.

/ID=nn
/INDEX=nn
Specifies the process for which information is to be displayed by its index into the system's list of software process control blocks (PCBs), or by its process identification (ID). You can supply the following values for nn:

• The process index itself
SDA Commands
SHOW PROCESS

- The process identification (PID) or extended PID longword, from which SDA extracts the correct index. The PID or extended PID of any thread of a process with multiple kernel threads may be specified. Any thread-specific data displayed by SHOW PROCESS will be for the given thread.

To obtain these values for any given process, issue the SDA command SHOW SUMMARY/THREADS. The /ID=nn and /INDEX=nn qualifiers can be used interchangeably.

/INVALID_PFN [=option]
See the /PAGE_TABLES qualifier description for an explanation of /INVALID_PFN.

/L1
/L2
/L3
When used with the /PAGE_TABLES qualifier, /L1, /L2, /L3 cause SDA to display the page table entries at the level specified. /L3 is the default.

/LOCKS [/BRIEF]
Displays the lock management locks owned by the current process.

The /LOCKS [/BRIEF] qualifier produces a display similar in format to that produced by the SHOW LOCK command. See also the /BRIEF qualifier description. Also, Table 4–4 contains additional information.

/NEXT
Causes SDA to locate the next valid process in the process list and select that process. If there are no further valid processes in the process list, SDA returns an error.

/NONMEMORY_PFN [=option]
See the /PAGE_TABLES qualifier description for an explanation of /NONMEMORY_PFN.

/P0
/P1
/P2
When used with the /PAGE_TABLES qualifier, /P0, /P1, /P2 cause SDA to display only page table entries for the specified region. /P0 is the default.

/PAGE_TABLES
The /PAGE_TABLES qualifier has the following format:

/PAGE_TABLES {range| [/P0(d)]| [/P1]| [/P2]| [/PT]}
| /GSTX=index /BDE=id
| /REGIONS=id
| /SECTION_INDEX=n| =ALL}
| /PT_ADDRESS
| [/INVALID_PFN [=READONLY|WRITABLE]]
| [/NONMEMORY_PFN [=READONLY|WRITABLE]]
| [/L1| [/L2]| [/L3(d)]}

Displays the page tables of the process P0 (process), P1 (control), P2, or PT (page table) region, or, optionally, page table entries for a range of addresses. The page table entries at the level specified by /L1, /L2, or /L3 (the default) are displayed.
When /RDE=id or /REGIONS=id is used with /PAGE_TABLES, SDA displays the page tables for the address range of the specified address region. When you do not specify an ID, the page tables are displayed for all the process-permanent and user-defined regions.

You can express a range using the following syntax:

m Displays the single page table entry that corresponds to virtual address m.
m:n Displays the page table entries that correspond to the range of virtual addresses from m to n.
m;n Displays the page table entries that correspond to a range of n bytes, starting at virtual address m.
=ALL Use /PAGE_TABLES=ALL to display the entire page table or the process from address zero to the end of process-private page table space.

The /PTE_ADDRESS qualifier causes SDA to treat the specified range as PTE addresses instead of virtual addresses.

The /SECTION_INDEX=n qualifier causes SDA to display only the page table entries for the pages in the specified process section.

The /GSTX=index qualifier causes SDA to display only the page table entries for the pages in the specified global section.

The /INVALID_PFN qualifier which is valid on platforms that supply an I/O memory map, causes SDA to display only page table entries that map to PFNs that are not in the system’s private memory, nor in Galaxy shared memory, nor are I/O access pages.

The /NONMEMORY_PFN qualifier, supported on all platforms, causes SDA to display only page table entries that are neither in the system’s private memory nor in Galaxy shared memory.

Both /INVALID_PFN and /NONMEMORY_PFN qualifiers allow two optional keywords, READONLY and WRITABLE. If neither keyword is given, all relevant pages are displayed. If you specify READONLY, only pages marked for no write access are displayed. If you specify WRITABLE, only pages that allow write access are displayed. For example, SHOW PROCESS ALL/PAGE_TABLE=ALL/INVALID_PFN=WRITABLE would display all process pages (for all processes) whose protection allows write, but which map to PFNs that do not belong to this system.

/PCB
Displays the information contained in the process control block (PCB). This is the default behavior of the SHOW PROCESS command.

/PERSO NA [=address]
Displays all persona security blocks (PSBs) held in the PERSONA ARRAY of the process, and then lists selected information contained in each initially listed PSB. The selected information includes the contents of the following cells inside the PSB:

Flags
Reference count
Execution mode
Audit status
Account name
UIC
SHOW PROCESS

Privileges
Rights enabled mask

If you specify a PSB address, the above information is provided for that specific PSB only.

/PERSONA/RIGHTS
Displays all the /PERSONA [=address] information and additional selected information, including all the Rights and their attributes currently held and active for each persona security block (PSB).

/PERSONA/RIGHTS/AUTHORIZED
Displays all the /PERSONA [=address] information and additional selected information, including all the Rights and their attributes authorized for each persona security block (PSB).

/PHD
Lists the information included in the process header (PHD).

/PPT
Is a synonym for /PAGE_TABLES.

/PROCESS_SECTION_TABLE [/SECTION_INDEX=id]
Lists the information contained in the process section table (PST). The /SECTION_INDEX=id qualifier used with /PROCESS_SECTION_TABLE displays the process section table entry for the specified section.

/PST
Is a synonym for /PROCESS_SECTION_TABLE.

/PT
When used with the /PAGE_TABLES qualifier, causes SDA to display the page table entries for the page table space of the process.

/PTE_ADDRESS
When used with the /PAGE_TABLES qualifier, specifies that the range is of PTE addresses instead of the virtual addresses mapped by the PTE.

/RDE [=id]
/REGIONS [=id]
Lists the information contained in the process region table for the specified region. If no region is specified, the entire table is displayed, including the process-permanent regions. The qualifiers /RDE [=id] and /REGIONS [=id] may be used interchangeably. When used with the /PAGE_TABLES, causes SDA to display on the page tables for the region given or all regions.

/REGISTERS
Lists the hardware context of the process, as reflected in the process registers stored in the hardware privileged context block (HWPCB), in its kernel stack, and possibly, in its PHD.

/RIGHTS
Used with the /PERSONA qualifier. See the /PERSONA/RIGHTS description for use of the /RIGHTS qualifier.
/RMS [=option[,...]]
Displays certain specified RMS data structures for each image I/O or process permanent I/O file the process has open. To display RMS data structures for process-permanent files, specify the PIO option to this qualifier.

SDA determines the structures to be displayed according to either of the following methods:

- If you provide the name of a structure or structures in the option parameter, SHOW PROCESS/RMS displays information from only the specified structures. (See Table 4-2 for a list of keywords that may be supplied as options.)
- If you do not specify an option, SHOW PROCESS/RMS displays the current list of options as shown by the SHOW_RMS command and set by the SET RMS command.

/SECTION_INDEX=n
When used with the /PAGE_TABLES qualifier, displays the page table for the range of pages in the specified process section. One of the qualifiers /L1, /L2, or /L3 can also be specified.

When used with the /PROCESS_SECTION_TABLE qualifier, displays the PST for the specified process section.

The /SECTION_INDEX=n qualifier is ignored if you do not specify either the /PAGE_TABLES or the /PROCESS_SECTION_TABLE qualifier.

/SEMAPHORE
Displays the Inner Mode Semaphore for a multithreaded process.

/SYSTEM
Displays the system process control block. Use of the process-name parameter, the /INDEX qualifier, or the /SYSTEM qualifier causes the SHOW PROCESS command to perform an implicit SET PROCESS command, making the indicated process the current process for subsequent SDA commands. (See the description of the SET PROCESS command and Section 2.5 for information on how this can affect the process context—and CPU context—in which SDA commands execute.) The system PCB and process header (PHD) parallel the data structures that describe processes. They contain the system working set, global section table, global page table, and other systemwide data.

/THREADS
Displays the software and hardware context of all the threads associated with the current process.

/WORKING_SET_LIST [={PPT|PROCESS|LOCKED|GLOBAL|MODIFIED|n}]}
Displays the contents of the requested entries of the working set list for the process. If you do not specify an option, then all working set list entries are displayed. Table 4-16 shows the options available with SHOW PROCESS/WORKING_SET_LIST.
Table 4–16 Options for the /WORKING_SET_LIST Qualifier

<table>
<thead>
<tr>
<th>Options</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPT</td>
<td>Displays process page table pages</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Displays process private pages</td>
</tr>
<tr>
<td>LOCKED</td>
<td>Displays pages locked into the process's working set</td>
</tr>
<tr>
<td>GLOBAL</td>
<td>Displays global pages currently in the working set of the process</td>
</tr>
<tr>
<td>MODIFIED</td>
<td>Displays working set list entries marked modified</td>
</tr>
<tr>
<td>n</td>
<td>Displays a specific working set list entry, where n is the working set list index (WSLX) of the entry of interest</td>
</tr>
</tbody>
</table>

Description

The SHOW PROCESS command displays information about the process specified by process-name, the process specified in the /INDEX qualifier, the system process, or all processes. The SHOW PROCESS command performs an implicit SET PROCESS command under certain uses of its qualifiers and parameters, as noted previously. By default, the SHOW PROCESS command produces information about the SDA current process, as defined in Section 2.5.

The default of the SHOW PROCESS command provides information taken from the software process control block (PCB) and the kernel threads block (KTB) of the SDA current thread. This is the first display provided by the /ALL qualifier and the only display provided by the /PCB qualifier. This information describes the following characteristics of the process:

- Software context
- Condition-handling information
- Information on interprocess communication
- Information on counts, quotas, and resource usage

Among the displayed information are the process PID, EPID, priority, job information block (JIB) address, and process header (PHD) address. SHOW PROCESS also describes the resources owned by the process, such as event flags and mutexes. The “State” field records current scheduling state for the thread, and indicates the CPU ID of any thread whose state is CUR.

The /THREADS qualifier (also part of SHOW PROCESS/ALL), displays information from the KTBs of all threads in the process, instead of only the SDA current thread.

The SHOW PROCESS/ALL command displays additional process-specific information, also provided by several of the individual qualifiers to the command.

The process registers display, also produced by the /REGISTERS qualifier, describes the process hardware context, as reflected in its registers. The registers displayed are those of the SDA current thread, or of all threads if either the /THREADS or the /ALL qualifier have been specified.

There are two places where a process hardware context is stored:

- If the process is currently executing on a processor in the Alpha system (that is, in the CUR scheduling state), its hardware context is contained in that processor’s registers. (That is, the process registers and the processor’s registers contain identical values, as illustrated by a SHOW CPU command.
for that processor or a SHOW CRASH command, if the process was current at
the time of the system failure).

- If the process is not executing, its privileged hardware context is stored in the
  part of the PHD known as the HWPCB. Its integer register context is stored
  on its kernel stack. Its floating-point registers are stored in its PHD.

The process registers display first lists those registers stored in the HWPCB,
kernel stack, and PHD ("Saved process registers"). If the process to be displayed
is currently executing on a processor in the Alpha system, the display then lists
the processor’s registers ("Active registers for the current process"). In each
section, the display lists the registers in the following groups:

- Integer registers (R0 through R29)
- Special-purpose registers (PC and PS)
- Stack pointers (KSP, ESP, SSP, and USP)
- Page table base register (PTBR)
- AST enable and summary registers (ASTEN and ASTSR)
- Address space number register (ASN)

The semaphore display, also produced by the /SEMAPHORE qualifier, provides
information on the inner-mode semaphore used to synchronize kernel threads.
The PC history log, recorded if the system parameter SYSTEM_CHECK is
enabled, is also displayed.

The process header display, also produced by the /PHD qualifier, provides
information taken from the PHD, which is swapped into memory when the
process becomes part of the balance set. Each item listed in the display reflects a
quantity, count, or limit for the process use of the following resources:

- Process memory
- The pager
- The scheduler
- Asynchronous system traps
- I/O activity
- CPU activity

The working set information and working set list displays, also produced by
the /WORKING_SET_LIST qualifier, describe those virtual pages that the process
can access without a page fault. After a brief description of the size, scope, and
characteristics of the working set list itself, SDA displays information for each
entry in the working set list as shown in Table 4–17.

Table 4–17 Working Set List Entry Information in the SHOW PROCESS Display

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>Index into the working set list at which information for this entry can be found</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>Virtual address of the page that this entry describes</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 4–17 (Cont.) Working Set List Entry Information in the SHOW PROCESS Display

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
<td>Four columns that list the following status information:</td>
</tr>
<tr>
<td></td>
<td>• Page status of VALID</td>
</tr>
<tr>
<td></td>
<td>• Type of physical page (See Table 4–9)</td>
</tr>
<tr>
<td></td>
<td>• Indication of whether the page has been modified</td>
</tr>
<tr>
<td></td>
<td>• Indication of whether the page is locked into the working set</td>
</tr>
</tbody>
</table>

When SDA locates either one or more unused working set entries, or entries that do not match the specified option, it issues the following message:

---- n entries not displayed

In this message, n is the number (in decimal) of contiguous entries not displayed.

The **process section table information** and **process section table** displays, also produced by the /PROCESS_SECTION_TABLE or /PST qualifier, list each entry in the process section table (PST) and display the offsets to, and the indexes of, the first free entry and last used entry.

SDA displays the information listed in Table 4–18 for each PST entry.

Table 4–18 Process Section Table Entry Information in the SHOW PROCESS Display

<table>
<thead>
<tr>
<th>Part</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
<td>Index number of the entry. Entries in the process section table begin at the highest location in the table, and the table expands toward lower addresses.</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>Address of the process section table entry.</td>
</tr>
<tr>
<td>SECTION ADDRESS</td>
<td>Virtual address that marks the beginning of the first page of the section described by this entry.</td>
</tr>
<tr>
<td>PAGELETS</td>
<td>Length of the process section. This is in units of pagelets, except for a PFN-mapped section in which the units are pages.</td>
</tr>
<tr>
<td>WINDOW</td>
<td>Address of the window control block on which the section file is open.</td>
</tr>
<tr>
<td>VBN</td>
<td>Virtual block number. The number of the file's virtual block that is mapped into the section's first page.</td>
</tr>
<tr>
<td>CCB</td>
<td>Address of the channel control block on which the section file is open.</td>
</tr>
<tr>
<td>REFCNT</td>
<td>Number of pages of this section that are currently mapped. (continued on next page)</td>
</tr>
</tbody>
</table>
In addition, for each process section that has an associated file, the device and/or filename is displayed. For details of this display, see Table 4–19.

The **regions** display, also produced by the either of the /RDE or /REGIONS qualifiers, shows the contents of the region descriptors. This includes the three default regions (P0, P1, P2), plus any others created by the process. A single region will be displayed if its identifier is specified. The information displayed for each region includes the RDE address, the address range of the region, its identifiers and protection, and links to other RDEs.

If the /PAGE_TABLE or /PPT qualifier is given with /RDE or /REGION, the page table for the region(s) are also displayed, as described below.

The **P0 page table, P1 page table, P2 page table, and PT page table** displays, also produced by the /PAGE_TABLES qualifier, display listings of the process page table entries in the same format as that produced by the SHOW PAGE_TABLE command (see Tables 4–5 through Table 4–10).

The **RMS** display, also produced by the /RMS qualifier, provides information on the RMS internal data structures for all RMS-accessed open files. The data structures displayed depend on the current setting of RMS options, as described under the SET RMS command and Table 4–2.

The **locks** display, also produced by the /LOCKS qualifier, provides information on the locks held by the process. For a full description of the information displayed for process locks, see the SHOW LOCKS command and Table 4–4. The /BRIEF qualifier may also be specified, giving a single line summary of each process lock; however, no other qualifiers from SHOW LOCKS apply to SHOW PROCESS/LOCKS.

The **process active channels** display, also produced by the /CHANNEL qualifier, displays the following information for each I/O channel assigned to the process:

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>Number of the channel</td>
</tr>
<tr>
<td>Window</td>
<td>Address of the window control block (WCB) for the file if the device is a file-oriented device; zero otherwise</td>
</tr>
<tr>
<td>Status</td>
<td>Status of the device: “Busy” if the device has an I/O operation outstanding; blank otherwise</td>
</tr>
<tr>
<td>Device/file accessed</td>
<td>Name of the device and, if applicable, name of the file being accessed on that device</td>
</tr>
</tbody>
</table>
The information listed under the heading “Device/file accessed” varies from channel to channel and from process to process. SDA displays certain information according to the conditions listed in Table 4–19.

Table 4–19 Process I/O Channel Information in the SHOW PROCESS Display

<table>
<thead>
<tr>
<th>Information Displayed</th>
<th>Type of Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcuu:</td>
<td>SDA displays this information for devices that are not file structured, such as terminals, and for processes that do not open files in the normal way.</td>
</tr>
<tr>
<td>dcuu:filespec</td>
<td>SDA displays this information only if you are examining a running system, and only if your process has enough privilege to translate the file-id into the filespec.</td>
</tr>
<tr>
<td>dcuu:(file-id)filespec</td>
<td>SDA displays this information only when you are examining a dump. The filespec corresponds to the file-id on the device listed. If you are examining a dump from your own system, the filespec is probably valid. If you are examining a dump from another system, the filespec is probably meaningless in the context of your system.</td>
</tr>
<tr>
<td>dcuu:(file-id)</td>
<td>The file-id no longer points to a valid filespec, as when you look at a dump from another system; or the process in which you are running SDA does not have enough privilege to translate the file-id into the corresponding filespec.</td>
</tr>
<tr>
<td>section file</td>
<td>The file in question is mapped into the process’s memory.</td>
</tr>
</tbody>
</table>

1This table uses the following conventions to identify the information displayed:
dcuu:(file-id)filespec
where:
dcuu: is the name of the device.
file-id is the RMS file identification.
filespec is the full file specification, including directory name.

The images display, also produced by the /IMAGES qualifier, describes the activated images in the process. SDA displays the information listed in Table 4–20 for each image, plus a summary line giving the total image and total page counts.

Table 4–20 Image Information in the SHOW PROCESS Display

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Name</td>
<td>The name of the image.</td>
</tr>
<tr>
<td>Link Time1</td>
<td>The date and time the image was linked.</td>
</tr>
<tr>
<td>Section Type1</td>
<td>For shareable images, the data for each image section is displayed on a separate line. For privileged shareable images, data for the change mode vector is also displayed on a separate line.</td>
</tr>
</tbody>
</table>

1These items are only displayed with SHOW PROCESS/IMAGE=ALL or SHOW PROCESS/ALL.

(continued on next page)
Table 4–20 (Cont.) Image Information in the SHOW PROCESS Display

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Start address of the image in process memory. For resident shareable images, this is the start address of the process-space portion of the image.</td>
</tr>
<tr>
<td>End</td>
<td>End address of the image in process memory. For resident shareable images, this is the end address of the process-space portion of the image.</td>
</tr>
<tr>
<td>Type</td>
<td>The image type and/or activation method, plus &quot;PROT&quot; for protected images and &quot;SHR&quot; for shareable images.</td>
</tr>
<tr>
<td>IMCB</td>
<td>The address of the Image Management Control Block.</td>
</tr>
<tr>
<td>Sym Vect&lt;sup&gt;1&lt;/sup&gt;</td>
<td>The address of the image's symbol vector, if any.</td>
</tr>
<tr>
<td>Maj,Minor ID&lt;sup&gt;1&lt;/sup&gt;</td>
<td>The major and minor revision IDs for the image.</td>
</tr>
<tr>
<td>Base&lt;sup&gt;1&lt;/sup&gt;</td>
<td>For shareable images, the base address of each image section and/or the change mode vector.</td>
</tr>
<tr>
<td>Fnd&lt;sup&gt;1&lt;/sup&gt;</td>
<td>For shareable images, the end address of each image section and/or the change mode vector.</td>
</tr>
<tr>
<td>ImageOff&lt;sup&gt;1&lt;/sup&gt;</td>
<td>For shareable images, the virtual offset within the image file for each image section.</td>
</tr>
</tbody>
</table>

<sup>1</sup>These items are only displayed with SHOW PROCESS/IMAGE=ALL or SHOW PROCESS/ALL.

The **buffer objects** display, also produced by the /BUFFER_OBJECTS qualifier, describes the buffer objects in use by the process. Information displayed by SDA for each buffer object includes its address, access mode, size, flags, plus the base virtual address of the object in process space and system space.

The **fast I/O handles** display, also produced by the /FANDLES qualifier, describes the fast I/O handles used by the process. Information displayed by SDA includes the address and size of the fast I/O handle vector header, then the address, corresponding IRP, state, and buffer object handles for each Fast I/O handle, plus information on free vector entries.

The **persona** display, also produced by the /PERSONA qualifier, describes the Persona status block data structures. The default output of /PERSONA consists of summary information for all personae in use by the process (the PSB address, flags, username) and information for each persona (privilege masks, UIC, and so on). When /PERSONA/RIGHTS is specified (as in SHOW PROCESS/ALL), all the rights currently held and active for each persona are also displayed. When /PERSONA/RIGHTS/AUTHORIZED is specified, all the rights authorized for each persona are displayed instead.
Examples

1. SDA> SHOW PROCESS
   Process index: 001A  Name: VERIFICATION  Extended PID: 0000051A  
   Process status:  22040023  RES,PHDRS,INTER
   status2:  00000001  QUANTUM_RESCHED

   PCB address  80613240  JIB address  805B1B40
   PHD address  80C3A000  Swapfile disk address  00000000
   KTB vector address  80D775AC  HWPCB address  81260080
   Callback vector address  00000000  Termination mailbox  0000
   Master internal PID  0005001A  Subprocess count  0
   Creator extended PID  00000000  Creator internal PID  00000000
   Previous CPU Id  00000000  Current CPU Id  00000000
   Previous ASNSEQ  0000000000000001  Previous ASN  000000000000002E
   Initial process priority  4  Delete pending count  0
   # open files allowed left  100  Direct I/O count/limit  150/150
   UIC  [00001,00004]  Buffered I/O count/limit  149/150
   Abs time of last event  005D9941  BUFIO byte count/limit  32128/32128
   ASTs remaining  247  # of threads  1
   Swapped copy of LEFC0  00000000  Timer entries allowed left  20
   Swapped copy of LEFC1  00000000  Active page table count  0
   Global cluster 2 pointer  00000000  Process WS page count  250
   Global cluster 3 pointer  00000000  Global WS page count  0

   Extended PID: 00000052  Thread index: 0000

   The SHOW PROCESS command displays information taken from the software PCB of VERIFICATION, the SDA current process. According to the “State” field in the display, process VERIFICATION is in Local Event Flag Wait.
2. SDA> SHOW PROCESS/ALL

Process index: 0013  Name: ACME_SERVER  Extended PID: 00000413

Process status: 00040011  RES,PSMAPM,PHDRES
status2: 00000010  TCB
PCB address 81500880  JIB address 815CE8A0
PHD address 83F62000  Swapfile disk address 00000000
KTB vector address 815CF840  HWPBC address 83F62080
Call-back vector address 815B8760  Termination mailbox 0000
Master internal PID 00010013  Subprocess count 0
Creator extended PID 00000000  Creator internal PID 00000000
Previous Id 00000000  Current CPU Id 8
Previous ASNSQ 0000000000000075  Previous ASN 00000000000000FD
Initial process priority 8  Delete pending count 0
# open files allowed left 97  Direct I/O count/limit 200/200
UIC [00001,000004]  Buffered I/O count/limit 199/200
Abs time of last event 0004BD6F  BUFIO byte count/limit 66272/66272
ASTs remaining 199  # of threads 2
Global cluster 2 pointer 00000000  Process WS page count 343
Global cluster 3 pointer 00000000  Global WS page count 101

Process index: 0013  Name: ACME_SERVER  Extended PID: 00000413

Thread index: 0000
---

Current capabilities: System: 00000000  QUORUM,RUN
User: 00000000
Permanent capabilities: System: 00000000  QUORUM,RUN
User: 00000000
Current affinities: 00000000
Permanent affinities: 00000000
Thread status: 00040011  status2: 00000010

KTB address 815D0880  HWPBC address 83F62080
PKTA address 7FFF0F98  Callback vector address 815B8780
Internal PID 00010013  Callback error 00000000
Extended PID 00000413  Current CPU id 00000000
State HIB Flags 00000000
Base priority 8  Current priority 13
Waiting EF cluster 0  Event flag wait mask 00130013
CPU since last quantum 0286  Mutex count 0
ASTs active NONE

Current process registers
---
R0 = 00000000.00000001  R1 = FFFFFFFF.81D0880  R2 = 00000000.7BC1CFF0
R3 = 00000000.7BC1CFF0  R4 = 00000000.0009D740  R5 = 00000000.7BC22E38
R6 = 00000000.00000000  R7 = 00000000.00000040  R8 = 00000000.00000001
R9 = 00000000.00000000  R10 = 00000000.00000060  R11 = 00000000.00000004
R12 = 00000000.00000000  R13 = 00000000.00000060  R14 = 00000000.00000001
R15 = 00000000.7BC65558  R16 = 00000000.00000000  R17 = 00000000.00000000
R18 = 00000000.00000000  R19 = 00000000.00000000  R20 = FFFFFFFF.FFFFFFFF
R21 = 00000000.00000000  R22 = 00000000.00000000  R23 = 00000000.00000000
R24 = 00000000.00000000  R25 = 00000000.00000000  R26 = FFFFFFFF.801270C8
R27 = FFFFFFFF.810D0B20  R28 = 00000000.00000000  FP = 00000000.00000000
PC = FFFFFFFF.80001934  PS = 00000000.00000000  SSP = 00000000.7F9A4000
KSP = 00000000.7F9A4000  ESP = 00000000.7F9A4000  FPCR = 00000000.00000000

continued
VM-0754AI
Thread index: 0001
------------------
Current capabilities:    System: 0000002C  QUORUM,RUN
User:   00000000
Permanent capabilities:  System: 0000002C  QUORUM,RUN
User:   00000000
Current affinities:      00000000
Permanent affinities:    00000000
Thread status:           00040011
status2:          00000010
KTB address              8153DA80    HWPCB address            84026200
PKTA address             40015F98    Callback vector address  815BB780
Internal PID 00020013    Callback error  00000000
Extended PID 0000813    Current CPU id  00000000
State HIB    Flags  00000000
Base priority     Current priority 13
Waiting SF cluster 0    Event flag wait mask 7FFFFFFF
CPU since last quantum 0036    Mutex count 0
ASPs active    NONE
Current process registers
-------------------------
R0   = 00000000.00000001  R1   = FFFFFFFF.815D0880  R2   = 00000000.7BC1CFF0
R3   = 00000000.7BC1CFF0  R4   = 00000000.000CB740  R5   = 00000000.7BC22E38
R6   = 00000000.00000080  R7   = 00000000.00000040  R8   = 00000000.00000001
R9   = 00000000.00000000  R10  = 00000000.00000000  R11  = 00000000.00000004
R12  = 00000000.00000000  R13  = FFFFFFFF.810D0B20  R14  = 00000000.7BC230B0
R15  = 00000000.7BC65558  R16  = 00000000.00000001  R17  = 00000000.000C9BE8
R18  = 00000000.00000000  R19  = 00000000.00000000  R20  = FFFFFFFF.FFFFFFFE
R21  = 00000000.00000006  R22  = 00000000.00000000  R23  = 00000000.00000001
R24  = 00000000.000C9BE8  R25  = 00000000.00000000  R26  = FFFFFFFF.801270C8
R27  = FFFFFFFF.810CD888  R28  = 00000000.00000006  FP   = 00000000.000C9C20
PC   = FFFFFFFF.80001934  PS   = 00000000.0000001B
KSP  = 00000000.40003EF0  ESP  = 00000000.40008000  SSP  = 00000000.4000C000
USP  = 00000000.000C9C20  PTBR = 00000000.00004F65
AST(SR/EN) = 00000000.00000000  ASN = 00000000.000000FF
F0   = 00000000.00000000  F1   = 00000000.00000000  F2   = 00000000.00000000
F3   = 00000000.00000000  F4   = 00000000.00000000  F5   = 00000000.00000000
F6   = 00000000.00000000  F7   = 00000000.00000000  F8   = 00000000.00000000
F9   = 00000000.00000000  F10  = 00000000.00000000  F11  = 00000000.00000000
F12  = 00000000.00000000  F13  = 00000000.00000000  F14  = 00000000.00000000
F15  = 00000000.00000000  F16  = 00000000.00000000  F17  = 00000000.00000000
F18  = 00000000.00000000  F19  = 00000000.00000000  F20  = 00000000.00000000
F21  = 00000000.00000000  F22  = 00000000.00000000  F23  = 00000000.00000000
F24  = 00000000.00000000  F25  = 00000000.00000000  F26  = 00000000.00000000
F27  = 00000000.00000000  F28  = 00000000.00000000  F29  = 00000000.00000000
F30  = 00000000.00000000  FPCR = 00000000.00000000
Process index: 0013   Name: ACME_SERVER   Extended PID: 00000413
----------------------------------------------------------------
Inner Mode Semaphore Address:   84026000
Owner:                              0000
Ownership Depth:                    0000
Tolerant count:                     0000
Flags:                              0000
History Buffer Is Empty
Process index: 0013   Name: ACME_SERVER   Extended PID: 00000413
----------------------------------------------------------------
Process header
--------------
First free P0 VA  00000000.00822000    Accumulated CPU time 00000004D
First free P1 VA  00000000.00822000    Subprocess quota 10
First free P2 VA  00000000.00822000    ASTs enabled KEU
First free P3 VA  00000000.00822000    ASTs enabled
Free page file pages 1565    ASN sequence # 0000000000000075
Page fault cluster size 4    AST limit 200
Page Table cluster size 1    Process header index 000D
Flags 0000000026    Backup address vector 0005C9A8
Direct I/O count 17    PTs having locked WSLEs 3
Page Table Base Register 00004F65    Maximum active PTs 8
Page Table Base register 00004F65    Maximum page file count 2500
Page Table Base register 00004F65    Guaranteed fluid WS pages 20
Page Table Base register 00004F65    File limit 1529
Page Table Base register 00004F65    Extra dynamic WS entries 1529
Page Table Base register 00004F65    Local event flag cluster 1 80000000
Page Table Base register 00004F65    Local event flag cluster 0 00000001
Page Table Base register 00004F65    Pagefile refcnt 00000000.000000F0
Page Table Base register 00004F65    Virtual PT Base FFFFFFFFC.00000000
continued
**SDA Commands**

**SHOW PROCESS**

---

**Process index:** 0013  **Name:** ACME_SERVER  **Extended PID:** 00000413

---

**Working set information**

---

First WSL entry: 00000001  Current authorized working set size: 3144
First locked entry: 00000009  Default (initial) working set size: 1572
First dynamic entry: 00000010  Maximum working set allowed (quota): 3144
Last entry replaced: 0000018C
Last entry in list: 0000056A

---

**Working set list**

---

<table>
<thead>
<tr>
<th>INDEX</th>
<th>ADDRESS</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>FFFFFEFD.BF6FC000</td>
<td>VALID PPT(L1) WSLOCK</td>
</tr>
<tr>
<td>00000002</td>
<td>FFFFFEFD.BF000000</td>
<td>VALID PPT(L2) WSLOCK</td>
</tr>
<tr>
<td>00000003</td>
<td>FFFFFEFC.001FE000</td>
<td>VALID PPT(L3) WSLOCK</td>
</tr>
<tr>
<td>00000004</td>
<td>00000000.7FAA0000</td>
<td>VALID PROCESS MODIFIED WSLOCK</td>
</tr>
<tr>
<td>00000005</td>
<td>00000000.7FF000000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
<tr>
<td>00000006</td>
<td>FFFFD7F.83F620000</td>
<td>VALID PHD WSLOCK</td>
</tr>
<tr>
<td>00000007</td>
<td>FFFFD7F.83F640000</td>
<td>VALID PHD WSLOCK</td>
</tr>
<tr>
<td>00000008</td>
<td>FFFFD7F.83F640000</td>
<td>VALID PHD WSLOCK</td>
</tr>
</tbody>
</table>

**Locked entries:**

<table>
<thead>
<tr>
<th>INDEX</th>
<th>ADDRESS</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000009</td>
<td>00000000.7AFE0000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
<tr>
<td>0000000A</td>
<td>00000000.7AFE2000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
<tr>
<td>0000000B</td>
<td>FFFFD7F.84026000</td>
<td>VALID PHD WSLOCK</td>
</tr>
<tr>
<td>0000000C</td>
<td>00000000.40002000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
<tr>
<td>0000000D</td>
<td>00000000.40002000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
<tr>
<td>0000000E</td>
<td>00000000.40014000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
<tr>
<td>0000000F</td>
<td>00000000.40016000</td>
<td>VALID PROCESS WSLOCK</td>
</tr>
</tbody>
</table>

**Dynamic entries:**

<table>
<thead>
<tr>
<th>INDEX</th>
<th>ADDRESS</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>00000000.7FFCE000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>00000011</td>
<td>FFFFD7F.001EA000</td>
<td>VALID PPT(L3) WSLOCK</td>
</tr>
<tr>
<td>00000012</td>
<td>00000000.7AFD0000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>00000013</td>
<td>00000000.7FBB8000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>00000014</td>
<td>00000000.7FDE0000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>00000015</td>
<td>00000007.7FD000000</td>
<td>VALID PROCESS MODIFIED</td>
</tr>
<tr>
<td>00000016</td>
<td>00000000.7FBAD000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000001B4</td>
<td>FFFFD7F.00020000</td>
<td>VALID PPT(L3) WSLOCK</td>
</tr>
<tr>
<td>000001B5</td>
<td>00000000.00860000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001B6</td>
<td>00000000.006F2000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001B7</td>
<td>00000000.006F4000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001B8</td>
<td>00000000.00804000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001B9</td>
<td>00000000.00818000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001BA</td>
<td>00000000.0080A000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001BB</td>
<td>00000000.0080C000</td>
<td>VALID PROCESS</td>
</tr>
<tr>
<td>000001BC</td>
<td>00000000.0081C000</td>
<td>VALID PROCESS</td>
</tr>
</tbody>
</table>

---- 1128 entries not displayed

---

**Process section table information**

---

Last entry allocated: 00000009
First free entry: 00000009

---

**continued**

VM-0756A-AI
### Process section table

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
<th>Section Address</th>
<th>Pagelets</th>
<th>Window</th>
<th>VBN</th>
<th>CCB</th>
<th>Refcnt</th>
<th>Flink</th>
<th>Blink</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000001</td>
<td>83F67FD8</td>
<td>00000000.00108000</td>
<td>0000005D</td>
<td>815BA440</td>
<td>00000004</td>
<td>7FEB8180</td>
<td>00000006</td>
<td>0008</td>
<td>0006</td>
<td>AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000002</td>
<td>83F67FB0</td>
<td>00000000.7B88C000</td>
<td>00000001</td>
<td>815BF840</td>
<td>00000003</td>
<td>7FEB8260</td>
<td>00000000</td>
<td>0005</td>
<td>0005</td>
<td>CRF WRT AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000003</td>
<td>83F67F88</td>
<td>00000000.00030000</td>
<td>000000A3</td>
<td>815C9440</td>
<td>0000002B</td>
<td>7FEB8020</td>
<td>0000000B</td>
<td>0004</td>
<td>0004</td>
<td>AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000004</td>
<td>83F67F60</td>
<td>00000000.00050000</td>
<td>00000001</td>
<td>815C9440</td>
<td>000000CE</td>
<td>7FEB8020</td>
<td>00000001</td>
<td>0003</td>
<td>0003</td>
<td>AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000005</td>
<td>83F67F38</td>
<td>00000000.7B91C000</td>
<td>00000013</td>
<td>815BF840</td>
<td>00000343</td>
<td>7FEB8260</td>
<td>00000000</td>
<td>0002</td>
<td>0002</td>
<td>CRF WRT AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000006</td>
<td>83F67F10</td>
<td>00000000.00148000</td>
<td>00000087</td>
<td>815BAA40</td>
<td>00000145</td>
<td>7FEB8180</td>
<td>00000000</td>
<td>0001</td>
<td>0001</td>
<td>CRF WRT AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000007</td>
<td>83F67E8</td>
<td>00000000.00168000</td>
<td>00000C23</td>
<td>815BAA40</td>
<td>000001CC</td>
<td>7FEB8180</td>
<td>000000C3</td>
<td>0006</td>
<td>0008</td>
<td>AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000008</td>
<td>83F67EC0</td>
<td>00000000.00378000</td>
<td>00000003</td>
<td>815BAA40</td>
<td>00000DF0</td>
<td>7FEB8180</td>
<td>00000001</td>
<td>0007</td>
<td>0007</td>
<td>AMOD=KRNL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000009</td>
<td>83F67E98</td>
<td>00000000.7B92C000</td>
<td>00000001</td>
<td>815BF840</td>
<td>00000356</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>0005</td>
<td>0002</td>
<td>CRF WRT AMOD=KRNL</td>
</tr>
</tbody>
</table>

---

### Process Region Table

<table>
<thead>
<tr>
<th>RDE Addr</th>
<th>Flink</th>
<th>Blink</th>
<th>T Link</th>
<th>Flags</th>
<th>Protect</th>
<th>Region Ident</th>
<th>Starting Address</th>
<th>Region Size</th>
<th>First Free VA</th>
</tr>
</thead>
<tbody>
<tr>
<td>7FEBA328</td>
<td>7FEBA328</td>
<td>7FEBA328</td>
<td>00000000</td>
<td>0000000A</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000.00000000</td>
<td>00000000.00000000</td>
<td>00000000.40000000</td>
</tr>
<tr>
<td>7FEBA360</td>
<td>7FE99960</td>
<td>7FE99960</td>
<td>00000000</td>
<td>0000001D</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000.00000001</td>
<td>00000000.40168000</td>
<td>00000000.7AF8E000</td>
</tr>
<tr>
<td>7FEBA398</td>
<td>7FEBA398</td>
<td>7FEBA398</td>
<td>00000000</td>
<td>00000008</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000.00000002</td>
<td>00000000.68000000</td>
<td>00000000.80000000</td>
</tr>
<tr>
<td>7FE99960</td>
<td>7FEBA360</td>
<td>7FEBA360</td>
<td>00000000</td>
<td>00000004</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000.00000003</td>
<td>00000000.40000000</td>
<td>00000000.40000000</td>
</tr>
</tbody>
</table>

---

### P0 space

<table>
<thead>
<tr>
<th>Mapped Address</th>
<th>PTE Address</th>
<th>PTE</th>
<th>Type</th>
<th>Read Writ</th>
<th>Bits</th>
<th>GHTyp</th>
<th>Loc</th>
<th>Bak</th>
<th>RefCnt</th>
<th>Flink</th>
<th>Blink</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>------</td>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>00000000.00000000</td>
<td>FFFFFFFF.00000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>t-to</td>
<td>00000000.00024000</td>
<td>t-to</td>
<td>FFFFFFFF.00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000.00000000</td>
<td>FFFFFFFF.00000040</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>t-to</td>
<td>00000000.00024000</td>
<td>t-to</td>
<td>FFFFFFFF.00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000.00000000</td>
<td>FFFFFFFF.00000008</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>t-to</td>
<td>00000000.00024000</td>
<td>t-to</td>
<td>FFFFFFFF.00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000.00000000</td>
<td>FFFFFFFF.00000080</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>t-to</td>
<td>00000000.00024000</td>
<td>t-to</td>
<td>FFFFFFFF.00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000.00000000</td>
<td>FFFFFFFF.00000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>t-to</td>
<td>00000000.00024000</td>
<td>t-to</td>
<td>FFFFFFFF.00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### P1 space

<table>
<thead>
<tr>
<th>Mapped Address</th>
<th>PTE Address</th>
<th>PTE</th>
<th>Type</th>
<th>Read</th>
<th>Writ</th>
<th>Bits</th>
<th>GH</th>
<th>PgTyp</th>
<th>Loc</th>
<th>Bak</th>
<th>RefCnt</th>
<th>Flink</th>
<th>B link</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000.40018000</td>
<td>FFFFFFFC.0010000</td>
<td>PTE</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>PTE</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>PTE</td>
<td>FFFFFFFC.0010000</td>
</tr>
<tr>
<td>00000000.40018000</td>
<td>FFFFFFFC.0010000</td>
<td>PTE</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>PTE</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>FFFFFFFC.0010000</td>
<td>00000000.40018000</td>
<td>PTE</td>
<td>FFFFFFFC.0010000</td>
</tr>
</tbody>
</table>

### P2 space

<table>
<thead>
<tr>
<th>Mapped Address</th>
<th>PTE Address</th>
<th>PTE</th>
<th>Type</th>
<th>Read</th>
<th>Writ</th>
<th>Bits</th>
<th>GH</th>
<th>PgTyp</th>
<th>Loc</th>
<th>Bak</th>
<th>RefCnt</th>
<th>Flink</th>
<th>B link</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFE.00000000</td>
<td>FFFFFFFF.00000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
</tr>
</tbody>
</table>

### PT space

<table>
<thead>
<tr>
<th>Mapped Address</th>
<th>PTE Address</th>
<th>PTE</th>
<th>Type</th>
<th>Read</th>
<th>Writ</th>
<th>Bits</th>
<th>GH</th>
<th>PgTyp</th>
<th>Loc</th>
<th>Bak</th>
<th>RefCnt</th>
<th>Flink</th>
<th>B link</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFFF.00000000</td>
<td>FFFFFFFF.00000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>FFFFFFFF.00000000</td>
<td>00000000.80000000</td>
<td>PTE</td>
<td>FFFFFFFF.00000000</td>
</tr>
</tbody>
</table>

---

**VM-0758A-AI**
SDA Commands
SHOW PROCESS

FFFEFFEC.001FE000 FFFFEFFD.BF0007F8 00004FAD.40001309 VALID KE-- -LK- 0 PPT(L3) ACTIVE FF000000.00000000 0001 0000000E 00000003

-------- 768 null pages: VA FFFFEFFC.00200000 PTK FFFFEFFD.BF000800
            -to- FFFFEFFD.007BFF77 -to- FFFFEFFD.BF001FF8

-------- 914432 entries not in memory: VA FFFFEFFC.00800000 PTK FFFFEFFD.BF002000
            -to- FFFFEFFD.BF6F8000 -to- FFFFEFFD.BF6FDBF8

FFFEFFD.BF000000 FFFFEFFD.BF6FC000 00004FAD.40001109 VALID KE-- -LK- -L- 0 PPT(L2) ACTIVE FF000000.00000000 0001 00000008 00000002

-------- 893 null pages: VA FFFFEFFD.BF002000 PTK FFFFEFFD.BF6FC008
            -to- FFFFEFFD.BF6F8000 -to- FFFFEFFD.BF6FDBF8

Process index: 0013 Name: ACME_SERVER Extended PID: 00000413

--------------------------
ASB Address: 7B02E000
--------------------------
LTP_POOL: 7B030800 IMPURE: 7FFD00C4
BIA: 00002600 9728.
BID: 00000032 50.
FP: 7FFA5118 7FFD00C4
SP: 7FFA5118 7FFD00C4
FLAGS: 00000000 PERSONA_ID: 2
SAVED_ID: 1
IO_OPERATION/OLD_FAB: 00000000
P4_PARM: 00000000
STL: 00018292
BNM: 00000010
STALL_STRUCT: 00000000
BMAST: 00000000
SUCAST: 00000000
FAB: 7FFD1000
STACK: 7B02F200
STXTP: 7B02E070
STXBO: 7B02F200
STXLEN: 00001190 4496.
MODE_OFFSET: 00000001
SAVED_ASB: 00000000
BUF: 00000008 ASY_THREAD,STALL_WITH_PERSONA

--------------------------
BDB Address: 7B028710
--------------------------
FLINK: 7B02726C BID: 0C 12.
FLGS: 0
USERS: 0000
CACHE.VAL:00 0.
BUFF_ID: 0000 0.
SIZE: 00000000 NMBR: 00000000
ADDR: 00000000 VBN: 00000000
VBNSEQNO: 00000000 WAIT: 00000000
KFL: 00000000 CURBUFADA:0000000000000FC00
SEL_VBN: 00000000 PRE_CCTL: 00
ASB: 00000000
ALLOC_ADDR: 00000000 BI_BDA: 00000000
ALLOC_SIZE: 0000 0 AI_BDA: 00000000
VALUES: 00000000 POST_CCTL: 00
ISOB: 00000000 WAIT_O_FLINK: 00000000
ISOB: 00000000 WAIT_O_BLINK: 00000000
REUSE_COUNT: 00000000 IDA_BAK_LEVEL: 00

Process index: 0013 Name: ACME_SERVER Extended PID: 00000413

continued

VM-0759A-AI

4–168  SDA Commands
Process active channels

<table>
<thead>
<tr>
<th>Channel</th>
<th>Window</th>
<th>Status</th>
<th>Device/file accessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>00000000</td>
<td></td>
<td>WFGLX0$DKB400:</td>
</tr>
<tr>
<td>0020</td>
<td>815C4440</td>
<td></td>
<td>[VMSCOMMON.SYSEXE]ACME_SERVER.EXE;1</td>
</tr>
<tr>
<td>0030</td>
<td>815C580</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]LIBRTOS.EXE;1 (section file)</td>
</tr>
<tr>
<td>0040</td>
<td>815BD200</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]LIBRTL.EXE;1 (section file)</td>
</tr>
<tr>
<td>0050</td>
<td>815BD800</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]LIBRTLSHRTL.EXE;1 (section file)</td>
</tr>
<tr>
<td>0060</td>
<td>815BF480</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]CMASTIS_SHR.EXE;1 (section file)</td>
</tr>
<tr>
<td>0070</td>
<td>815C0980</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]DECSC$SRH.EXE;1 (section file)</td>
</tr>
<tr>
<td>0080</td>
<td>815C5500</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]DVMLSHR.EXE;1 (section file)</td>
</tr>
<tr>
<td>0090</td>
<td>815C7240</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSMRG]SHRMNGMSG.EXE;1 (section file)</td>
</tr>
<tr>
<td>00A0</td>
<td>815C6980</td>
<td>Busy</td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]VMS$VMS_ACMESHR.EXE;1</td>
</tr>
<tr>
<td>00B0</td>
<td>00000000</td>
<td></td>
<td>WFGLX0$DKB400:[SYS12.SYSMGR]ACME$SERVER.LOG;21</td>
</tr>
<tr>
<td>00C0</td>
<td>815D2480</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]VMS$VMS_ACMESHR.EXE;1</td>
</tr>
<tr>
<td>00D0</td>
<td>815BAD40</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]SECURESHR.EXE;1 (section file)</td>
</tr>
<tr>
<td>00E0</td>
<td>815BF1C0</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]SECURESHRP.EXE;1 (section file)</td>
</tr>
<tr>
<td>00F0</td>
<td>815BDF40</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]PTD$SERVICES_SHR.EXE;1 (section file)</td>
</tr>
<tr>
<td>0100</td>
<td>815BC440</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]CRFSHR.EXE;1 (section file)</td>
</tr>
<tr>
<td>0110</td>
<td>815BF640</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]TRACE.EXE;1 (section file)</td>
</tr>
<tr>
<td>0120</td>
<td>815C2140</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]ADARTL.EXE;1 (section file)</td>
</tr>
<tr>
<td>0130</td>
<td>815C0D40</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]VMS$VMS_ACMESHR.EXE;1</td>
</tr>
<tr>
<td>0140</td>
<td>815BF240</td>
<td></td>
<td>WFGLX0$DKB400:[VMSCOMMON.SYSLIB]VMS$VMS_ACMESHR.EXE;1</td>
</tr>
</tbody>
</table>

Total number of open channels : 20.

Process index: 0013   Name: ACME_SERVER   Extended PID: 00000413

Process activated images

<table>
<thead>
<tr>
<th>Image Name/Link Time/Section Type</th>
<th>Start</th>
<th>End</th>
<th>Type</th>
<th>IMCB</th>
<th>Sym Vect Major,Minor ID</th>
<th>Base</th>
<th>End</th>
<th>Image Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME_SERVER</td>
<td>00010000</td>
<td>000705FF</td>
<td>MAIN</td>
<td>7FE99860</td>
<td>113,1238597</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHRMNGMSG</td>
<td>000B4000</td>
<td>000B49FF</td>
<td>MRGD</td>
<td>7FE99840</td>
<td>000B4000 113,12524133</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECSSMSG</td>
<td>000B0C00</td>
<td>000B0FFFF</td>
<td>MRGD</td>
<td>7FE998A30</td>
<td>000B0C00 113,12609585</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMS$VMS_ACMESHR</td>
<td>00108000</td>
<td>0010FFFF</td>
<td>MRGD</td>
<td>7FE9992A0</td>
<td>00108200 113,12563930</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Resident Code</td>
<td>80800000</td>
<td>808027FF</td>
<td>00030000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable Address Data</td>
<td>7B2B4000</td>
<td>7B2B69FF</td>
<td>00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable Code</td>
<td>7B2B4000</td>
<td>7B2B59FF</td>
<td>00010000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable Read-Only Data</td>
<td>7B314000</td>
<td>7B314737</td>
<td>00060000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable Address Data</td>
<td>7B314000</td>
<td>7B314737</td>
<td>00060000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable Code</td>
<td>7B314000</td>
<td>7B314737</td>
<td>00060000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareable Read-Only Data</td>
<td>7B314000</td>
<td>7B314737</td>
<td>00060000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read-Write Data</td>
<td>7B314000</td>
<td>7B314737</td>
<td>00060000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compressed Data</td>
<td>7B334000</td>
<td>7B3348FF</td>
<td>00080000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS$PUBLIC_VECTORS</td>
<td>81003E78</td>
<td>81005E37</td>
<td>GLBL</td>
<td>7FE98840</td>
<td>81003E78 113,12237288</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS$BASE_IMAGE</td>
<td>81019D90</td>
<td>81021C23</td>
<td>GLBL</td>
<td>7FE98720</td>
<td>81019D90 113,12239366</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total images = 19   Pages allocated = 885
The SHOW PROCESS/ALL command displays information taken from the PCB and KTBs of process ACME_SERVER, and then proceeds to display the process registers, inner mode semaphores, the process header and working set, the process section table, process regions, the page tables of the process, RMS data structures, information about I/O channels owned by the process, images activated by the process, and process persona data structures. These displays may also be obtained using the /PCB, /THREADS, /REGISTERS, /SEMAPHORE, /PHD, /WORKING_SET_LIST, /PST, /RDE, /PAGE=ALL, /RMS, /CHANNELS, /IMAGES=ALL, and PERSONA/RIGHTS qualifiers, respectively. This process had no locks, buffer objects or fast I/O handles to be displayed.
SDA Commands

SHOW PROCESS/PAGE_TABLES/ADDRESS=805E7980

This example displays the page tables of a process whose PCB address is 805E7980.

4. SDA> SHOW PROCESS/BUFFER_OBJECTS/FANDLEs

Process index: 0022 Name: Milord_RTA1: Extended PID: 00000062

Process Buffer Objects

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>ACMODE</th>
<th>SEQUENCE</th>
<th>REFCNT</th>
<th>PID</th>
<th>PAGCNT</th>
<th>BASE PVA</th>
<th>BASE SVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8151AE00</td>
<td>User</td>
<td>00000011</td>
<td>00000031</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.00084000</td>
<td>FFFFFFFF.7DE68000 S2_WINDOW</td>
</tr>
<tr>
<td>814A6CC0</td>
<td>User</td>
<td>00000012</td>
<td>00000009</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.80000000</td>
<td>FFFFFFFF.7DE66000 S2_WINDOW</td>
</tr>
<tr>
<td>814FBA0</td>
<td>User</td>
<td>00000013</td>
<td>00000009</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.80000000</td>
<td>FFFFFFFF.FFFFFFFF NOSVA</td>
</tr>
<tr>
<td>81512200</td>
<td>User</td>
<td>00000014</td>
<td>00000009</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.80028000</td>
<td>FFFFFFFF.7DE64000 S2_WINDOW</td>
</tr>
<tr>
<td>8151A8C0</td>
<td>User</td>
<td>00000015</td>
<td>00000009</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.80028000</td>
<td>FFFFFFFF.FFFFFFFF NOSVA</td>
</tr>
<tr>
<td>81438580</td>
<td>User</td>
<td>00000016</td>
<td>00000009</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.80028000</td>
<td>FFFFFFFF.7DE62000 S2_WINDOW</td>
</tr>
<tr>
<td>81464480</td>
<td>User</td>
<td>00000017</td>
<td>00000009</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.80028000</td>
<td>FFFFFFFF.FFFFFFFF NOSVA</td>
</tr>
<tr>
<td>81416F00</td>
<td>Kernel</td>
<td>00000018</td>
<td>00000001</td>
<td>00100022</td>
<td>00000001</td>
<td>00000000.7F760000</td>
<td>FFFFFFF.F8120C000 NOQUOTA</td>
</tr>
</tbody>
</table>

Fandle Vector Header

Address  Maxfix  Real_Size  CCB buffer handle
7FF68290  00000043  00000880  00000018.81416F00

Fandles

<table>
<thead>
<tr>
<th>Address</th>
<th>IRP</th>
<th>fastio_done</th>
<th>Orgfun</th>
<th>Data bo handle</th>
<th>IOSA bo handle</th>
<th>DBYLEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>7FF68280</td>
<td>815CE40</td>
<td>set</td>
<td>00020031</td>
<td>00000016.81438580</td>
<td>00000011.8151AE00</td>
<td>00000000.00002000</td>
</tr>
<tr>
<td>7FF682D0</td>
<td>815CE40</td>
<td>set</td>
<td>00020030</td>
<td>00000016.81438580</td>
<td>00000011.8151AE00</td>
<td>00000000.00002000</td>
</tr>
<tr>
<td>7FF682F0</td>
<td>815CE20</td>
<td>set</td>
<td>00020031</td>
<td>00000016.81438580</td>
<td>00000011.8151AE00</td>
<td>00000000.00002000</td>
</tr>
<tr>
<td>7FF68310</td>
<td>815D4B80</td>
<td>set</td>
<td>00020031</td>
<td>00000015.8151A8C0</td>
<td>00000011.8151AE00</td>
<td>00000000.00002000</td>
</tr>
<tr>
<td>7FF68330</td>
<td>815D65C0</td>
<td>set</td>
<td>00020030</td>
<td>00000015.8151A8C0</td>
<td>00000011.8151AE00</td>
<td>00000000.00002000</td>
</tr>
</tbody>
</table>
The SHOW PROCESS/BUFFER_OBJECTS/FANCES command displays all the buffered objects and fast I/O handles that a process has created.
SHOW RAD

Displays the settings and explanations of the RAD_SUPPORT system parameter fields, and the assignment of CPUs and memory to the Resource Affinity Domains (RADs). This command is only useful on platforms that support RADs. By default, the SHOW RAD command displays the settings of the RAD_SUPPORT system parameter fields.

Format

SHOW RAD [number | /ALL]

Parameter

number
Displays information on CPUs and memory for the specified RAD.

Qualifier

/ALL
Displays settings of the RAD_SUPPORT parameter fields and the CPU and memory assignments for all RADs.

Examples

1. SDA> SHOW RAD

   Resource Affinity Domains
   --------------------------
   RAD information header address: FFFFFFFE.81032340
   Maximum RAD count: 00000008
   RAD containing SYS$BASE_IMAGE: 00000000
   RAD support flags: 0000004F
   +-----------+-----------+-----------+-----------+|..|..| 0| 0| 0| 0| 0|..|..|..|..|.1|00|11|11|+-----------+-----------+-----------+-----------+
   Bit 0 = 1: RAD support is enabled
   Bit 1 = 1: Soft RAD affinity support is enabled
               (Default scheduler skip count of 16 attempts)
   Bit 2 = 1: System-space replication support is enabled
   Bit 3 = 1: Copy on soft fault is enabled
   Bit 4 = 0: Default RAD-based page allocation in use
   Allocation Type  RAD choice
     ------------  ----------
     Process-private pagefault  Home
     Process creation or inswap  Random
     Global pagefault  Random
     System-space page allocation  Current
   Bit 5 = 0: RAD debug feature is disabled
Bit 6 = 1: Per-RAD non-paged pool is enabled

This example shows the settings of the RAD_SUPPORT system parameter fields.

2. SDA> SHOW RAD 2

Resource Affinity Domain 0002

CPU sets:
- Active 08 09 10 11
- Configure 08 09 10 11
- Potential 08 09 10 11

PFN ranges:

<table>
<thead>
<tr>
<th>Start PFN</th>
<th>End PFN</th>
<th>PFN count</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000000</td>
<td>0101FFFF</td>
<td>00020000</td>
<td>000A OpenVMS Base</td>
</tr>
<tr>
<td>01020000</td>
<td>0103FFFF</td>
<td>00020000</td>
<td>0010 Galaxy_Shared</td>
</tr>
</tbody>
</table>

SYSPTBR: 01003C00

This example shows information on the CPUs and memory for RAD 2.
SHOW RESOURCES

Displays information about all resources in the system, or about a resource associated with a specific lock.

Format

SHOW RESOURCES {/ADDRESS=n|/ALL (d)| /BRIEF | /CACHED | /CONTENTION [=ALL] | /LOCKID=lock-id | /NAME=resource-name | /OWNED | /STATUS=(keyword [keyword;])}

Parameters

None.

Qualifiers

/ADDRESS=n
Displays information from the resource block at the specified address.

/ALL
Displays information from all resource blocks (RSBs) in the system. This is the default behavior of the SHOW RESOURCES command.

/BRIEF
Displays a single line of information for each resource.

/CACHED
Displays resource blocks that are no longer valid. The memory for these resources is saved so that later requests for resources can use them.

/CONTENTION [=ALL]
Displays only resources that have at least one lock on either the waiting or conversion queue. Unless you specify the ALL keyword, resources with locks on the waiting or conversion queues that are not participating in deadlock searches are ignored. (Locks not participating in deadlock searches are requested with either the LCK$M_NODLCKWT or LCK$M_NODLCKBLK flags.)

/LOCKID=lock-id
Displays information on the resource associated with the lock with the specified lock-id.

/NAME=resource-name
Displays information about a specific resource.

/OWNED
Causes SDA to display only owned resources.

/STATUS=(keyword [keyword;])
Displays only resources that have the specified status bits set in the RSB$L_STATUS field. Status keywords are as follows:
### Key Definitions

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2PC_IP</td>
<td>Indicates a two-phase convert operation in progress</td>
</tr>
<tr>
<td>BRL</td>
<td>Indicates byte range resource</td>
</tr>
<tr>
<td>CHK_BTR</td>
<td>Checks for better master</td>
</tr>
<tr>
<td>CVTFULRNG</td>
<td>Indicates full-range requests in convert queue</td>
</tr>
<tr>
<td>CVTSUBRNG</td>
<td>Indicates sub-range requests in convert queue</td>
</tr>
<tr>
<td>DIRENTRY</td>
<td>Indicates entered in directory during failover</td>
</tr>
<tr>
<td>DIR_IP</td>
<td>Creates directory entry</td>
</tr>
<tr>
<td>DIR_RQD</td>
<td>Indicates directory entry required</td>
</tr>
<tr>
<td>INVPEND</td>
<td>Checks for value block invalidation</td>
</tr>
<tr>
<td>RBLD_ACT</td>
<td>Indicates lock rebuild active for this tree</td>
</tr>
<tr>
<td>RBLD_IP</td>
<td>Indicates rebuild operation in progress</td>
</tr>
<tr>
<td>RBLD_RQD</td>
<td>Indicates rebuild required for this resource tree</td>
</tr>
<tr>
<td>RM_ACCEPT</td>
<td>Accepts new master</td>
</tr>
<tr>
<td>RM_DEFLECT</td>
<td>Deflects remote interest</td>
</tr>
<tr>
<td>RM_IP</td>
<td>Indicates resource remaster in progress</td>
</tr>
<tr>
<td>RM_PEND</td>
<td>Indicates a pending resource remaster operation</td>
</tr>
<tr>
<td>RM_RBLD</td>
<td>Indicates to always rebuild resource tree</td>
</tr>
<tr>
<td>RM_WAIT</td>
<td>Blocks local activity</td>
</tr>
<tr>
<td>VALCUR</td>
<td>Indicates value block is current</td>
</tr>
<tr>
<td>VALINVLD</td>
<td>Indicates value block invalid</td>
</tr>
<tr>
<td>WTFULRNG</td>
<td>Indicates full-range requests in wait queue</td>
</tr>
<tr>
<td>WTSUBRNG</td>
<td>Indicates a full-range requests in wait queue</td>
</tr>
</tbody>
</table>

### Description

The SHOW RESOURCES command displays the information listed in Table 4–21 either for each resource in the system or for the specific resource associated with the specified **lock-id**, address or name.

### Table 4–21 Resource Information in the SHOW RESOURCES Display

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address of RSB</td>
<td>Address of the resource block (RSB) that describes this resource.</td>
</tr>
<tr>
<td>GG MODE</td>
<td>Indication of the most restrictive mode in which a lock on this resource has been granted. Table 4–22 shows the fields and values and their meanings. They are shown in order from the least restrictive mode to the most restrictive.</td>
</tr>
<tr>
<td>Status</td>
<td>The contents of the resource block status field.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 4–21 (Cont.) Resource Information in the SHOW RESOURCES Display

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent RSB</td>
<td>Address of the RSB that is the parent of this RSB. This field is 00000000 if the RSB itself is a parent block.</td>
</tr>
<tr>
<td>CGMODE</td>
<td>Indication of the most restrictive lock mode to which a lock on this resource is waiting to be converted. This does not include the mode for which the lock at the head of the conversion queue is waiting. See Table 4–22.</td>
</tr>
<tr>
<td>Sub-RSB count</td>
<td>Number of RSBs of which this RSB is the parent. This field is 0 if the RSB has no sub-RSBs.</td>
</tr>
<tr>
<td>FGMODE</td>
<td>Indication of the full-range grant mode. See Table 4–22.</td>
</tr>
<tr>
<td>Lock Count</td>
<td>The total count of all locks on the resource.</td>
</tr>
<tr>
<td>RQSEQNM</td>
<td>Sequence number of the request.</td>
</tr>
<tr>
<td>BLKAST count</td>
<td>Number of locks on this resource that have requested a blocking AST.</td>
</tr>
<tr>
<td>CSID</td>
<td>Cluster system identification number (CSID) and name of the node that owns the resource.</td>
</tr>
<tr>
<td>Resource</td>
<td>Dump of the name of this resource, as stored at the end of the RSB. The first two columns are the hexadecimal representation of the name, with the least significant byte represented by the rightmost two digits in the rightmost column. The third column contains the ASCII representation of the name, the least significant byte being represented by the leftmost character in the column. Periods in this column represent values that correspond to nonprinting ASCII characters.</td>
</tr>
<tr>
<td>Valblk</td>
<td>Hexadecimal dump of the 16-byte block value block associated with this resource.</td>
</tr>
<tr>
<td>Length</td>
<td>Length in bytes of the resource name.</td>
</tr>
<tr>
<td>Mode</td>
<td>Processor mode of the namespace in which this RSB resides.</td>
</tr>
<tr>
<td>Owner</td>
<td>Owner of the resource. Certain resources, owned by the operating system, list “System” as the owner. Locks owned by a group have the number (in octal) of the owning group in this field.</td>
</tr>
<tr>
<td>Seqnum</td>
<td>Sequence number associated with the resource's value block. If the number indicates that the value block is not valid, the words “Not valid” appear to the right of the number.</td>
</tr>
<tr>
<td>Granted queue</td>
<td>List of locks on this resource that have been granted. For each lock in the list, SDA displays the number of the lock and the lock mode in which the lock was granted.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 4–21 (Cont.) Resource Information in the SHOW RESOURCES Display

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion queue</td>
<td>List of locks waiting to be converted from one mode to another. For each lock in the list, SDA displays the number of the lock, the mode in which the lock was granted, and the mode to which the lock is to be converted.</td>
</tr>
<tr>
<td>Waiting queue</td>
<td>List of locks waiting to be granted. For each lock in the list, SDA displays the number of the lock and the mode requested for that lock.</td>
</tr>
</tbody>
</table>

Table 4–22 Lock on Resources

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>Null mode</td>
</tr>
<tr>
<td>CR</td>
<td>Concurrent-read mode</td>
</tr>
<tr>
<td>CW</td>
<td>Concurrent-write mode</td>
</tr>
<tr>
<td>PR</td>
<td>Protected-read mode</td>
</tr>
<tr>
<td>PW</td>
<td>Protected-write mode</td>
</tr>
<tr>
<td>EX</td>
<td>Exclusive mode</td>
</tr>
</tbody>
</table>

Examples

1. SDA> SHOW RESOURCES

Resource database

RSB: FFFFFFFF.7FD47950 GGMODE: PR Status: VALID
Parent RSB: 00000000.00000000 CGMODE: PR
Sub-RSB count: 0 FGMODE: PR
Lock Count: 1 RQSEQNM: 0000
BLKAST count: 1 CSID: 00000000 (SWORKS)

Resource: 6D632445 48434143 CACHE$cm Valblk: 00000000 00000000
Length 24 525F534B 524F5753 SWORKS_R 00000000 00000000
Kernel mode 000027DA 4E455641 AVENU’... System 00000000 00000000 ........ Seqnum: 00000000

Granted queue (Lock ID / Gr mode / Range):
0100042F PR 00000000-FFFFFFFF

Conversion queue (Lock ID / Gr mode / Range -> Rq mode / Range):
*** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode / Range):
*** EMPTY QUEUE ***

Resource Database

RSB: FFFFFFFF.7FA66A50 GGMODE: NL Status: VALID
Parent RSB: FFFFFFFF.7FD88350 CGMODE: NL
Sub-RSB count: 0 FGMODE: NL
Lock Count: 2 RQSEQNM: 004D
BLKAST count: 0 CSID: 00000000 (SWORKS)
The SHOW RESOURCES command displays information taken from the RSBs of all resources in the system. For instance, the RSB at FFFFFFFF.7FA66A50 is a parent block with no sub-RSBs.

This example of the SHOW RESOURCES/CONTENTION commands shows all the resources for which there is contention, and which are not to be included in dead lock searches.
SHOW RMD

Displays information contained in the reserved memory descriptors. Reserved memory is used within the system by memory-resident global sections.

Format

SHOW RMD [/QUALIFIERS]

Parameters

None.

Qualifiers

/ADDRESS=\nDisplays a specific reserved memory descriptor entry, given its address.

/ALL
Displays information in all the reserved memory descriptors. This qualifier is the default.

Description

The SHOW RMD displays information that resides in the reserved memory descriptors. Table 4–23 shows the fields and their meaning.

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>Gives the address of the reserved memory descriptor.</td>
</tr>
<tr>
<td>NAME</td>
<td>Gives the name of the reserved memory descriptor.</td>
</tr>
<tr>
<td>GROUP</td>
<td>Gives the UIC group that owns the reserved memory. This is given as -S- for system global reserved memory.</td>
</tr>
<tr>
<td>RAD</td>
<td>Gives the required RAD for the reserved memory. Displays &quot;Any&quot; if no RAD specified.</td>
</tr>
<tr>
<td>PFN</td>
<td>Gives starting page number of the reserved memory.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Gives the number of pages reserved.</td>
</tr>
<tr>
<td>IN_USE</td>
<td>Gives the number of pages in use. If an error occurred when the reserved memory was being allocated, the error condition code is displayed in parentheses. A second line, giving the text of the error, is also displayed in this case.</td>
</tr>
<tr>
<td>/ERROR</td>
<td></td>
</tr>
<tr>
<td>ZERO_PFN</td>
<td>Gives the next page number to be zeroed.</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Gives the settings of flags for specified reserved memory descriptor, as a hexadecimal number, then key flag bits are also displayed by name. The names may use multiple lines in the display.</td>
</tr>
</tbody>
</table>
### Example

SDA> SHOW RMD

Reserved Memory Descriptor List

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Group RAD</th>
<th>PFN</th>
<th>Count</th>
<th>In_Use (Error)</th>
<th>Zero_PFN</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>814199C0</td>
<td>LARGE</td>
<td>00022</td>
<td>00000000</td>
<td>000004E2</td>
<td>00000000</td>
<td>00000000E0</td>
<td>Group Page_Tables GBLsec</td>
</tr>
<tr>
<td>81419940</td>
<td>LARGE</td>
<td>00022</td>
<td>00000000</td>
<td>00138800</td>
<td>(0000244C)</td>
<td>00000000</td>
<td>Error Group GBLsec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Error = %SYSTEM-F-INSFLPGS, insufficient Fluid Pages available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81419AC0</td>
<td>SMALL</td>
<td>00011</td>
<td>00000180</td>
<td>00000001</td>
<td>00000000</td>
<td>00000180</td>
<td>000000E1 Alloc Group Page_Tables GBLsec</td>
</tr>
<tr>
<td>81419A40</td>
<td>SMALL</td>
<td>00011</td>
<td>000001E00</td>
<td>00000080</td>
<td>00000000</td>
<td>00000E00</td>
<td>00000A1 Alloc Group GBLsec</td>
</tr>
</tbody>
</table>
SHOW RMS

Displays the RMS data structures selected by the SET RMS command to be included in the default display of the SHOW PROCESS/RMS command.

Format

SHOW RMS

Parameters

None.

Qualifiers

None.

Description

The SHOW RMS command lists the names of the data structures selected for the default display of the SHOW PROCESS/RMS command.

For a description of the significance of the options listed in the SHOW RMS display, see the description of the SET RMS command and Table 4–2.

For an illustration of the information displayed by the SHOW PROCESS/RMS command, see the examples included in the description of the SHOW PROCESS command.

Examples

1. SDA> SHOW RMS

   RMS Display Options: IFB, IRB, IDX, BDB, BDBSUM, ASB, CCB, WCB, FCB, FAB, RAB, NAM, XAB, RLB, BLB, BLBSUM, GBD, GBH, FWA, GBDSUM, JFB, NWA, RU, DRC, SFSB, GBSB

   Display RMS structures for all IFI values.

   The SHOW RMS command displays the full set of options available for display by the SHOW PROCESS/RMS command. SDA, by default, selects the full set of RMS options at the beginning of an analysis.

2. SDA> SET RMS=(IFAB=1, CCB, WCB)
   SDA> SHOW RMS

   RMS Display Options: IFB, CCB, WCB

   Display RMS structures only for IFI =0001

   The SET RMS command establishes the IFB, CCB, and WCB as the structures to be displayed, and only for the file whose internal File Identifier has the value 1, when the SHOW PROCESS/RMS command is issued. The SHOW RMS command verifies this selection of RMS options.
SHOW RSPID

Displays information about response IDs (RSPIDs) of all System Communications Services (SCS) connections or, optionally, a specific SCS connection.

Format

SHOW RSPID [/CONNECTION=cdt-address]

Parameters

None.

Qualifier

/CONNECTION=cdt-address
Displays RSPID information for the specific SCS connection whose connection descriptor table (CDT) address is provided in cdt-address. You can find the cdt-address for any active connection on the system in the CDT summary page display of the SHOW CONNECTIONS command. CDT addresses are also stored in many individual data structures related to SCS connections. These data structures include class driver request packets (CDRPs) and unit control blocks (UCBs) for class drivers that use SCS and cluster system blocks (CSBs) for the connection manager.

Description

Whenever a local system application (SYSAP) requires a response from a remote SYSAP, a unique number, called an RSPID, is assigned to the response by the local system. The RSPID is transmitted in the original request (as a means of identification), and the remote SYSAP returns the same RSPID in its response to the original request.

The SHOW RSPID command displays information taken from the response descriptor table (RDT), which lists the currently open local requests that require responses from SYSAPs at a remote node. For each RSPID, SDA displays the following information:

- RSPID value
- Address of the class driver request packet (CDRP), which generally represents the original request
- Address of the CDT that is using the RSPID
- Name of the local process using the RSPID
- Remote node from which a response is required (and has not yet been received)
Examples

1. SDA> SHOW RSPID

--- Summary of Response Descriptor Table (RDT) 805E6F18 ---

<table>
<thead>
<tr>
<th>RSPID</th>
<th>CDRP Address</th>
<th>CDT Address</th>
<th>Local Process Name</th>
<th>Remote Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>39D00000</td>
<td>8062CC80</td>
<td>805E8710</td>
<td>VMS$VMScluster</td>
<td>VANDQ1</td>
</tr>
<tr>
<td>EE210001</td>
<td>80637260</td>
<td>805E8C90</td>
<td>VMS$DISK_CL_DRVR</td>
<td>ROMRDR</td>
</tr>
<tr>
<td>EE240002</td>
<td>806382E0</td>
<td>805E8DF0</td>
<td>VMS$DISK_CL_DRVR</td>
<td>VANDQ1</td>
</tr>
<tr>
<td>EE440003</td>
<td>806393E0</td>
<td>805E8F50</td>
<td>VMS$TAPE_CL_DRVR</td>
<td>VANDQ1</td>
</tr>
<tr>
<td>5DB90004</td>
<td>80636BC0</td>
<td>805E8870</td>
<td>VMS$VMScluster</td>
<td>ROMRDR</td>
</tr>
<tr>
<td>5C260005</td>
<td>80664040</td>
<td>805E8B70</td>
<td>VMS$VMScluster</td>
<td>ROMRDR</td>
</tr>
<tr>
<td>38F80006</td>
<td>80664A80</td>
<td>805E8710</td>
<td>VMS$VMScluster</td>
<td>VANDQ1</td>
</tr>
</tbody>
</table>

This example shows the default output for the SHOW RSPID command.

2. SDA> SHOW RSPID/CONNECTION=805E8F50

--- Summary of Response Descriptor Table (RDT) 805E6F18 ---

<table>
<thead>
<tr>
<th>RSPID</th>
<th>CDRP Address</th>
<th>CDT Address</th>
<th>Local Process Name</th>
<th>Remote Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE440003</td>
<td>806393E0</td>
<td>805E8F50</td>
<td>VMS$TAPE_CL_DRVR</td>
<td>VANDQ1</td>
</tr>
</tbody>
</table>

This example shows the output for a SHOW RSPID/CONNECTION command.
SHOW SHM_CPP

Displays information about the shared memory common property partitions (CPPs). The default display shows a single page summary which includes a single line for each CPP.

Format

SHOW SHM_CPP [/ QUALIFIERS]

Parameters

None.

Qualifiers

/ADDRESS=n
Displays a detailed page of information about an individual shared memory CPP given the address of the SHM_CPP structure.

/ALL
Displays a detailed page of information about each shared memory CPP.

/IDENT=n
Displays a detailed page of information about an individual shared memory CPP.

/PFN [=option]
Displays PFN data in addition to the basic SHM_CPP. The default is all lists (free, bad, untested), plus the PFN database pages and the complete range of PFNs in the CPP.

To display only the complete range of PFNs in the CPP, use the keyword ALL_FRAGMENTS with the /PFN qualifier:

/PFN = ALL_FRAGMENTS

To display only the bad page list, use the keyword BAD with the /PFN qualifier:

/PFN = BAD

To display only the free page list, use the keyword FREE with the /PFN qualifier:

/PFN = FREE

To display the PFNs containing the PFN database, use the keyword PFNDB with the /PFN qualifier:

/PFN = PFNDB

To display only the untested page list, use the keyword UNTESTED with the /PFN qualifier:

/PFN = UNTESTED

To display multiple lists, you can combine keywords with the /PFN qualifier:

/PFN = (x,y)

Note that if /PFN is given without /ALL, /IDENT, or /ADDRESS, then the system displays the PFN list(s) from the last shared memory CPP accessed.
SDA Commands
SHOW SHM_CPP

Examples

1. SDA> SHOW SHM_CPP
   Summary of Shared Memory Common Property Partitions
   -----------------------------------------------
   Base address of SHM_CPP array: FFFFFFFF.7F2BA140
   Maximum number of SHM_CPP entries: 00000007
   Size of each SHM_CPP: 000000240
   Maximum fragment count per SHM_CPP: 00000010
   Valid CPP count: 00000001
<table>
<thead>
<tr>
<th>ID</th>
<th>SHM_CPP address</th>
<th>MinPFN</th>
<th>MaxPFN</th>
<th>Page count</th>
<th>Free pages</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- SHM_CPP IDs 0000 to 0002: VALID flag clear --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>FFFFFFF.7F2BA800</td>
<td>00060000</td>
<td>0007FFFF</td>
<td>00020000</td>
<td>0001FCF7</td>
<td>00000001</td>
</tr>
<tr>
<td>-- SHM_CPP IDs 0004 to 0006: VALID flag clear --</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
   
   This example shows the default output for the SHOW SHM_CPP command.

2. SDA> SHOW SHM_CPP/IDENT=3
   Shared Memory CPP 0003
   -----------------------------
   SHM_CPP address: FFFFFFFF.7F2BA800
   Version: 00000001 Flags: 00000001 VALID
   Size: 00000000.000000C0 Page count: 00020000
   Actual fragment count: 00000001 Minimum PFN: 00060000
   Maximum fragment count: 00000010 Maximum PFN: 0007FFFF
   Length of free page list: 0001FCF7
   Length of bad page list: 00000000
   Length of untested page list: 00000000
   PMAP array for PFN database pages
   PMAP | Start PFN | PFN count
   ------|----------|----------
   0. | 00660053 | 00000280
   PMAP array for all fragments
   PMAP | Start PFN | PFN count
   ------|----------|----------
   0. | 00660000 | 00020000
   GLock address: FFFFFFFF.7F2BA8C0 Handle: 00000000.00010D19
   GLock name: SHM_CPP00000003 Flags: 00
   Owner count: 00 Owner node: 00
   Node sequence: 00 Owner: 000000
   IPL: 08 Previous IPL: 00
   Wait bitmask: 00000000.00000000 Timeout: 00249F00
   Thread ID: 00000000.00000000
   Connected GNode bitmask: FFFFFFFF.7F2BA900
   Valid bits: 0000004 State: 00000000.00000000
   Unit count: 0001 Unit size: QUADWORD
   Unit bitmask: ........ ........ ........ ........ 7 00000000
   Ranges of free pages
This example shows the details for a single SHM_CPP.

<table>
<thead>
<tr>
<th>Range</th>
<th>Start PFN</th>
<th>PFN count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>000602F6</td>
<td>00000002</td>
</tr>
<tr>
<td>2.</td>
<td>0006030B</td>
<td>0001FCF5</td>
</tr>
</tbody>
</table>
SHOW SHM_REG

Displays information about shared memory regions. The default display shows a single page summary which includes a single line for each region.

Format

SHOW SHM_REG [/ QUALIFIERS] [name]

Parameter

name
Displays a detailed page of information about the named region.

Qualifiers

/ADDRESS=n
Displays a detailed page of information about an individual region given the address of the SHM_REG structure.

/ALL
Displays a detailed page of information about each region.

/IDENT=n
Displays a detailed page of information about the specified region.

Examples

1. SDA>SHOW SHM_REG
   Summary of Shared Memory Regions
   --------------------------------------------------------------
   Base address of SHM_REG array: FFFFFFFF.7F2BB140
   Maximum number of SHM_REG entries: 00000040
   Size of each SHM_REG: 00000208
   Base address of SHM_DESC array: FFFFFFFF.7F2DC000
   Valid region count: 00000009
   -------------------------------------
   Region Tag SysVA / GSTX  Flags
   ------- ----------------- --------------------- ----- --------
   0000 FFFFFFFF.7F2BB140 SYS$GALAXY_MANAGEMENT_DATABASE FFFFFFFF.7F234000 00000001 VALID
   0001 FFFFFFFF.7F2BB348 SYS$SHARED_MEMORY_PFN_DATABASE FFFFFFFE.00000000 00000001 VALID
   0002 FFFFFFFF.7F2BB550 SMCI$SECTION_PBA_04001 -<None>- 00000001 VALID
   0003 FFFFFFFF.7F2BB758 GLX$CPU$BALANCER$SYSGBL 0000013F 00000005 VALID SHARED_CONTEXT_VALID
   0004 FFFFFFFF.7F2BB960 SMCI$CHANNEL_PBA_0_1 FFFFFFFF.8F3AE000 00000001 VALID
   0005 FFFFFFFF.7F2BBB68 SMCI$CHANNEL_PBA_0_2 FFFFFFFF.8FAE3000 00000001 VALID
   0006 FFFFFFFF.7F2BCD70 SMCI$CHANNEL_PBA_1_2 -<Not Attached>- 00000001 VALID
   0007 FFFFFFFF.7F2BC878 LAN$SHM_REG FFFFFFFF.7F2DC000 00000009 VALID ATTACH_DETACH
   0008 FFFFFFFF.7F2BC180 GLX$CPU_BAL_GLOCK $0000006 00000140 00000005 VALID SHARED_CONTEXT_VALID
   --------------------------------------------------------------
   -- SHM_REG IDs 0009 to 003F: never used --

   This example shows the summary of all shared memory regions in the system.

2. SDA> SHOW SHM_REG SMCI$CHANNEL_PBA_0_1
   --------------------------------------------------------------
   SHM_REG address: FFFFFFFF.7F2BB960
   Version: 00000001 Flags: 00000001 VALID
   Index/Sequence: 0004/00000003 Size: 00000000.00000120
   Region tag: SMCI$CHANNEL_PBA_0_1
   Creation time: 31-MAR-1999 14:11:11.37
   SHM_DESC address: FFFFFFFF.7F2DC200

   --------------------------------------------------------------
This example shows the details for a single shared memory region.
SHOW SPINLOCKS

Displays the multiprocessing synchronization data structures.

Format

SHOW SPINLOCKS  {[name] | /ADDRESS=expression | /INDEX=expression}
                [/COUNTS | /OWNED | /DYNAMIC | /STATIC]  {[/BRIEF | /FULL]}

Parameter

name
Name of the spinlock, fork lock, or device lock structure to be displayed. Device lock names are of the form [node$]lock, where node optionally indicates the OpenVMS Cluster node name (allocation class) and lock indicates the device and controller identification (for example, HAETAR$DUA).

Qualifiers

/ADDRESS=expression
Displays the lock at the address specified in expression. You can use the /ADDRESS qualifier to display a specific device lock; however, the name of the device lock is listed as “Unknown” in the display.

/BRIEF
Produces a condensed display of the lock information displayed by default by the SHOW SPINLOCKS command, including the following: address, spinlock name or device name, IPL or device IPL, rank, ownership depth, and CPU ID of the owner CPU. If the system under analysis was executing with full-checking multiprocessing enabled (according to the setting of the MULTIPROCESSING or SYSTEM_CHECK system parameter), then the number of waiting CPUs and interlock status are also displayed.

/COUNTS
Produces a display of Spin, Wait, and Acquire counts for each spinlock (only if full-checking multiprocessing enabled).

/DYNAMIC
Displays information for all device locks in the system.

/FULL
Displays full descriptive and diagnostic information for each displayed spinlock, fork lock, or device lock.

/INDEX=expression
Displays the system spinlock whose index is specified in expression. You cannot use the /INDEX qualifier to display a device lock.

/OWNED
Displays information for all spinlocks, fork locks, and device locks owned by the SDA current CPU. If a processor does not own any spinlocks, SDA displays the following message:

No spinlocks currently owned by CPU xx

The xx represents the CPU ID of the processor.
Displays information for all system spinlocks and fork locks.

**Description**

The SHOW SPINLOCKS command displays status and diagnostic information about the multiprocessing synchronization structures known as spinlocks.

A **static spinlock** is a spinlock whose data structure is permanently assembled into the system. Static spinlocks are accessed as indexes into a vector of longword addresses called the spinlock vector, the address of which is contained in SMP$AR_SPNLKVEC. System spinlocks and fork locks are static spinlocks. Table 4–24 lists the static spinlocks.

A **dynamic spinlock** is a spinlock that is created based on the configuration of a particular system. One such dynamic spinlock is the device lock SYSMAN creates when configuring a particular device. This device lock synchronizes access to the device's registers and certain UCB fields. The system creates a dynamic spinlock by allocating space from nonpaged pool, rather than assembling the lock into the system as it does in creating a static spinlock.

See the Writing OpenVMS Alpha Device Drivers in C for a full discussion of the role of spinlocks in maintaining synchronization of kernel mode activities in a multiprocessing environment.

**Table 4–24 Static Spinlocks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUEAST</td>
<td>Fork lock for queuing ASTs at IPL 6</td>
</tr>
<tr>
<td>FILSYS</td>
<td>Lock on file system structures</td>
</tr>
<tr>
<td>LCKMGR</td>
<td>Lock on all lock manager structures</td>
</tr>
<tr>
<td>IOLOCK8/SCS</td>
<td>Fork lock for executing a driver fork process at IPL 8</td>
</tr>
<tr>
<td>TX_SYNCH</td>
<td>Transaction processing lock</td>
</tr>
<tr>
<td>TIMER</td>
<td>Lock for adding and deleting timer queue entries and searching the timer queue</td>
</tr>
<tr>
<td>PORT</td>
<td>Template structure for dynamic spinlocks for ports with multiple devices</td>
</tr>
<tr>
<td>IO_MISC</td>
<td>Miscellaneous short term I/O locks</td>
</tr>
<tr>
<td>MMG</td>
<td>Lock on memory management, PFN database, swapper, modified page writer, and creation of per-CPU database structures</td>
</tr>
<tr>
<td>SCHED</td>
<td>Lock on process control blocks (PCBs), scheduler database, and mutex acquisition and release structures</td>
</tr>
<tr>
<td>IOLOCK9</td>
<td>Fork lock for executing a driver fork process at IPL 9</td>
</tr>
<tr>
<td>IOLOCK10</td>
<td>Fork lock for executing a driver fork process at IPL 10</td>
</tr>
<tr>
<td>IOLOCK11</td>
<td>Fork lock for executing a driver fork process at IPL 11</td>
</tr>
<tr>
<td>MAILBOX</td>
<td>Lock for sending messages to mailboxes</td>
</tr>
<tr>
<td>POOL</td>
<td>Lock on nonpaged pool database</td>
</tr>
<tr>
<td>PERFMON</td>
<td>Lock for I/O performance monitoring</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 4–24 (Cont.)  Static Spinlocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVALIDATE</td>
<td>Lock for system space translation buffer (TB) invalidation</td>
</tr>
<tr>
<td>HWCLK</td>
<td>Lock on hardware clock database, including the quadword containing the due</td>
</tr>
<tr>
<td></td>
<td>time of the first timer queue entry (EXE$GQ_1ST_TIME) and the quadword</td>
</tr>
<tr>
<td></td>
<td>containing the system time (EXE$GQ_SYS_TIME)</td>
</tr>
<tr>
<td>MEGA</td>
<td>Lock for serializing access to fork-wait queue</td>
</tr>
<tr>
<td>EMB/MCHECK</td>
<td>Lock for allocating and releasing error-logging buffers and synchronizing</td>
</tr>
<tr>
<td></td>
<td>certain machine error handling</td>
</tr>
</tbody>
</table>

For each spinlock, fork lock, or device lock in the system, SHOW SPINLOCKS provides the following information:

- Name of the spinlock (or device name for the device lock)
- Address of the spinlock data structure (SPL)
- The owning CPU’s CPU ID
- IPL at which allocation of the lock is synchronized on a local processor
- Number of nested acquisitions of the spinlock by the processor owning the spinlock (“Ownership Depth”)
- Rank of the spinlock
- Timeout interval for spinlock acquisition (in terms of 10 milliseconds)
- Shared array (shared spinlock context block pointer)
- Number of processors waiting to obtain the spinlock
- Interlock (synchronization mutex used when full-checking multiprocessing is enabled)

The last two items (CPUs waiting and Interlock) are only displayed if full-checking multiprocessing is enabled.

SHOW SPINLOCKS/BRIEF produces a condensed display of this same information, excluding the share array and timeout interval.

SHOW SPINLOCKS/COUNTS displays only the Spin, Wait, and Acquire counts for each spinlock.

If the system under analysis was executing with full-checking multiprocessing enabled, SHOW SPINLOCKS/FULL adds to the spinlock display the Spin, Wait, and Acquire counts and the last sixteen PCs at which the lock was acquired or released. If applicable, SDA also displays the PC of the last release of multiple, nested acquisitions of the lock.

If no spinlock name, address, or index is given, then information is displayed for all applicable spinlocks.
### Examples

1. **SDA> SHOW SPINLOCKS**

   System static spinlock structures
   ----------------------------------
   | Address | 810AE300 |
   | Owner CPU ID | None |
   | IPL | 0000001F |
   | Ownership Depth | FFFFFFFF |
   | Rank | 00000000 |
   | Timeout Interval | 000186A0 |
   | Share Array | 00000000 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   MCHECK
   | Address | 810AE300 |
   | Owner CPU ID | None |
   | IPL | 0000001F |
   | Ownership Depth | FFFFFFFF |
   | Rank | 00000000 |
   | Timeout Interval | 000186A0 |
   | Share Array | 00000000 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   MEGA
   | Address | 810AE400 |
   | Owner CPU ID | None |
   | IPL | 0000001F |
   | Ownership Depth | FFFFFFFF |
   | Rank | 00000000 |
   | Timeout Interval | 000186A0 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   HWCLK
   | Address | 810AE500 |
   | Owner CPU ID | None |
   | IPL | 00000016 |
   | Ownership Depth | FFFFFFFF |
   | Rank | 00000004 |
   | Timeout Interval | 000186A0 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   System dynamic spinlock structures
   ----------------------------------
   | Address | 8103FB00 |
   | Owner CPU ID | None |
   | DIPL | 00000015 |
   | Ownership Depth | FFFFFFFF |
   | Share Array | 00000000 |
   | Timeout Interval | 000186A0 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   QTV14$MB
   | Address | 810AE900 |
   | Owner CPU ID | None |
   | IPL | 0000000B |
   | Ownership Depth | FFFFFFFF |
   | Rank | 0000000C |
   | Timeout Interval | 000186A0 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   QTV14$NL
   | Address | 810AE900 |
   | Owner CPU ID | None |
   | IPL | 0000000B |
   | Ownership Depth | FFFFFFFF |
   | Rank | 0000000C |
   | Timeout Interval | 000186A0 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

   QTV14$PKA
   | Address | 814AA100 |
   | Owner CPU ID | None |
   | DIPL | 00000015 |
   | Ownership Depth | FFFFFFFF |
   | Share Array | 00000000 |
   | Timeout Interval | 000186A0 |
   | CPUs Waiting | 00000000 |
   | Interlock | Free |

This excerpt illustrates the default output of the SHOW SPINLOCKS command.
2. SDA> SHOW SPINLOCKS/BRIEF

System static spinlock structures
---------------------------------

<table>
<thead>
<tr>
<th>Spinlock</th>
<th>Address</th>
<th>IPL</th>
<th>Rank</th>
<th>Depth</th>
<th>Owner</th>
<th>CPU Waiting</th>
<th>Interlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMB</td>
<td>810AE300</td>
<td>001F</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>MCHECK</td>
<td>810AE300</td>
<td>001F</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>MEGA</td>
<td>810AE400</td>
<td>001F</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>HWCLK</td>
<td>810AE500</td>
<td>0016</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>INVALIDATE</td>
<td>810AE600</td>
<td>0015</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>PERFMON</td>
<td>810AE700</td>
<td>000F</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>POOL</td>
<td>810AE800</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>MAILBOX</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>IOLock11</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>IOLock10</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>IOLock9</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>SCHED</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>MMG</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>IOLOCK1</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>SCS</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>LCKMGR</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>FILSYS</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>QUEUEAST</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
</tbody>
</table>

System dynamic spinlock structures
-----------------------------------

<table>
<thead>
<tr>
<th>Device</th>
<th>Address</th>
<th>DIPL</th>
<th>Rank</th>
<th>Depth</th>
<th>Owner</th>
<th>CPUs Waiting</th>
<th>Interlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>QTV14OPA</td>
<td>8103FB00</td>
<td>0015</td>
<td>FFFFFFFF</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>QTV14MBA</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>QTV14NLA</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
<tr>
<td>QTV14PKA</td>
<td>810AE900</td>
<td>000B</td>
<td>00000000</td>
<td>FFFFFFFF</td>
<td>None</td>
<td>00000000</td>
<td>Free</td>
</tr>
</tbody>
</table>

This excerpt illustrates the condensed form of the display produced in the first example.

3. SDA> SHOW SPINLOCKS/FULL SCHED

System static spinlock structures
---------------------------------

<table>
<thead>
<tr>
<th>SPLSC_SCHED</th>
<th>Address</th>
<th>Owner CPU ID</th>
<th>IPL</th>
<th>Ownership Depth</th>
<th>Timeout Interval</th>
<th>CPUs Waiting</th>
<th>Spins</th>
<th>Acquires</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>810AED00</td>
<td>000000000</td>
<td>I</td>
<td>0000000000</td>
<td>002DC6C0</td>
<td>000000000</td>
<td></td>
<td>0000000000.0458EBDC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td>00000000000</td>
<td>00000000000</td>
<td>00000000000</td>
<td></td>
<td>0000000000.01279BE0</td>
</tr>
</tbody>
</table>
Spinlock SPL$C_SCHED was last acquired or released from:

(Most recently)  
8004AD00 EXE$SWTIMER_FORK_C+00170
. 8004B1D4 EXE$SWTIMER_FORK_C+00644
. 8004AD00 EXE$SWTIMER_FORK_C+00170
. 8004B1D4 EXE$SWTIMER_FORK_C+00644
. 8004AD00 EXE$SWTIMER_FORK_C+00170
. 8004B1D4 EXE$SWTIMER_FORK_C+00644
. 8004AD00 EXE$SWTIMER_FORK_C+00170
. 8004B1D4 EXE$SWTIMER_FORK_C+00644
. 8004AD00 EXE$SWTIMER_FORK_C+00170
. 8004B1D4 EXE$SWTIMER_FORK_C+00644
. 8004AD00 EXE$SWTIMER_FORK_C+00170
. 80136A2C SCH$INTERRUPT+0070C
. 80117580 SCH$IDLE_C+002A0
. 8004B230 EXE$SWTIMER_FORK_C+006A0
. 8004AFC4 EXE$SWTIMER_FORK_C+00434
. 80117360 SCH$IDLE_C+00080
. 8012E5F4 EXESHIBER_INT_C+00074

(Least recently)
80132150 EXE$SCHDWK_C+00110

Last release of multiple acquisitions occurred at:
80262A54 EXE$CHECK_VERSION_C+009F4

This display shows the detailed information on the SCHED spinlock, including the PC history.
**SHOW STACK**

Displays the location and contents of the process stacks (of the SDA current process) and the system stack.

**Format**

```plaintext
SHOW STACK  {range | /ALL | /EXECUTIVE | /INTERRUPT | /KERNEL
             | /PHYSICAL | /SUPERVISOR | /SYSTEM | /USER} {/LONG | /QUAD (d)}
```

**Parameter**

`range`

Range of memory locations you want to display in stack format. You can express a `range` using the following syntax:

- `m:n` Range of addresses from `m` to `n`
- `m;n` Range of addresses starting at `m` and continuing for `n` bytes

**Qualifiers**

`/ALL`

Displays the locations and contents of the four process stacks for the current SDA process and the system stack.

`/EXECUTIVE`

Shows the executive stack for the SDA current process.

`/INTERRUPT`

Shows the system stack and is retained for compatibility with OpenVMS VAX. The interrupt stack does not exist in OpenVMS Alpha.

`/KERNEL`

Shows the kernel stack for the SDA current process.

`/LONG`

Displays longword width stacks. If this qualifier is not specified, SDA by default displays quadword width stacks.

`/PHYSICAL`

Treats the start and/or end addresses in the given range as physical addresses. This qualifier is only relevant when a range is specified. By default, SDA treats range addresses as virtual addresses.

`/QUAD`

Displays quadword width stacks. This is the default.

`/SUPERVISOR`

Shows the supervisor stack for the SDA current process.

`/SYSTEM`

Shows the system stack.

`/USER`

Shows the user stack for the SDA current process.
Description

The SHOW STACK command, by default, displays the stack that was in use when the system failed, or, in the analysis of a running system, the current operating stack. For a process that became the SDA current process as the result of a SET PROCESS command, the SHOW STACK command by default shows its current operating stack.

The various qualifiers to the command allow display of any of the four per-process stacks for the SDA current process, as well as the system stack for the SDA current CPU. In addition, any given range can be displayed in stack format.

You can define SDA process and CPU context by using the SET CPU, SHOW CPU, SHOW CRASH, SET PROCESS, and SHOW PROCESS commands as indicated in their command descriptions. A complete discussion of SDA context control appears in Chapter 2, Section 2.5.

SDA provides the following information in each stack display:

<table>
<thead>
<tr>
<th>Section</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity of stack</td>
<td>SDA indicates whether the stack is a process stack (user, supervisor, executive, or kernel) or the system stack.</td>
</tr>
<tr>
<td>Stack pointer</td>
<td>The stack pointer identifies the top of the stack. The display indicates the stack pointer by the symbol SP =&gt;.</td>
</tr>
<tr>
<td>Stack address</td>
<td>SDA lists all the addresses that the operating system has allocated to the stack. The stack addresses are listed in a column that increases in increments of 8 bytes (one quadword), unless you specify the /LONG qualifier in which case addresses are listed in increments of 4 (one longword).</td>
</tr>
<tr>
<td>Stack contents</td>
<td>SDA lists the contents of the stack in a column to the right of the stack addresses.</td>
</tr>
<tr>
<td>Symbols</td>
<td>SDA attempts to display the contents of a location symbolically, using a symbol and an offset. If the stack is being displayed in quadword width and the location cannot be symbolized as a quadword, SDA will attempt to symbolize the least significant longword and then the most significant longword. If the address cannot be symbolized, this column is left blank.</td>
</tr>
<tr>
<td>Canonical stack</td>
<td>When displaying the kernel stack of a noncurrent process in a crash dump, SDA identifies the stack locations used by the scheduler to store the register contents of the process.</td>
</tr>
<tr>
<td>Mechanism array</td>
<td>When displaying the current stack in a FATALEXCPT, INVEXCEPTN, SSRVEXCEPT, or UNXSIGNAL bugcheck, SDA identifies the stack locations used to store registers and other key data for these structures.</td>
</tr>
<tr>
<td>Signal array</td>
<td></td>
</tr>
<tr>
<td>Exception frame</td>
<td></td>
</tr>
</tbody>
</table>

If a stack is empty, the display shows the following:

SP => (STACK IS EMPTY)
SDA Commands
SHOW STACK

Example

SDA> SHOW STACK
Current Operating Stack (SYSTEM):

```
FF...8244BD08  FFFF...800600FC  SCH$REPORT_EVENT_C+000FC
FF...8244BD10  00000000.00000002
FF...8244BD18  00000000.00000005
FF...8244BD20  FFFF...8060C7C0

SP => FF...8244BD28  FFFF...8244BE8

FF...8244BD30  FFFF...80018960  EXE$HWCLKINT_C+00260
FF...8244BD38  00000000.000001B8
FF...8244BD48  00000000.00000210  UCB$N_RSID+00002
FF...8244BD50  00000000.00000000
FF...8244BD58  00000000.00000000
FF...8244BD60  FFFF...804045D0  SCH$GQ_IDLE_CPUS
FF...8244BD68  FFFF...8041A340  EXE$GL_FKWAITFL+00020
FF...8244BD70  00000000.00000025  UCB$T_MSGDATA+00034
FF...8244BD78  00000000.00000001

CHF$IS_MCH_ARGS  FFFF...8244BD80  00000000.0000002B
CHF$PH_MCH_FRAME  FFFF...8244BD88  FFFF...8244BF80
CHF$IS_MCH_DEPTH  FFFF...8244BD90  80000000.FFFFFD  G
CHF$PH_MCH_ADDR  FFFF...8244BD98  00000000.0001600  CTLSC_CLIDATASZ+00060
CHF$PH_MCH_ESF_ADDR  FFFF...8244BDA0  FFFF...8244BF40
CHF$PH_MCH_SIG_ADDR  FFFF...8244BDA8  FFFF...8244BE8

CHF$IH_MCH_SAVR0  FFFF...8244BD00  FFFF...8041FB00  SMP$RELEASE+00640
CHF$IH_MCH_SAVR1  FFFF...8244BD08  00000000.00000000
CHF$IH_MCH_SAVR16  FFFF...8244BD0C  00000000.0000000D
CHF$IH_MCH_SAVR17  FFFF...8244BD08  00000000.00000000
CHF$IH_MCH_SAVR18  FFFF...8244BD00  00000000.00000000
CHF$IH_MCH_SAVR19  FFFF...8244BD08  00000000.00000001
CHF$IH_MCH_SAVR20  FFFF...8244BD00  00000000.00000000
CHF$IH_MCH_SAVR21  FFFF...8244BD08  FFFF...805AE4B6  SISR+0006E
CHF$IH_MCH_SAVR22  FFFF...8244BD00  00000000.00000001
CHF$IH_MCH_SAVR23  FFFF...8244BD08  00000000.00000010
CHF$IH_MCH_SAVR24  FFFF...8244BD00  00000000.00000008
CHF$IH_MCH_SAVR25  FFFF...8244BD00  00000000.00000010
CHF$IH_MCH_SAVR26  FFFF...8244BD10  00000000.00000001
CHF$IH_MCH_SAVR27  FFFF...8244BD18  00000000.00000000
CHF$IH_MCH_SAVR28  FFFF...8244BD20  FFFF...804045D0  SCH$GQ_IDLE_CPUS

```

The SHOW STACK command displays a system stack. The data shown above the stack pointer may not be valid. Note that the mechanism array, signal array, and exception frame symbols displayed on the left will appear only for INVEXCEPTN, FATALEXCPT, UNXSIGNAL, and SSRVEXCEPT bugchecks.
SHOW SUMMARY

Displays a list of all active processes and the values of the parameters used in swapping and scheduling these processes.

Format

SHOW SUMMARY [ /IMAGE | /PROCESS_NAME=process_name \\
/THREAD | /USER=username ]

Parameters

None.

Qualifiers

/IMAGE
Causes SDA to display, if possible, the name of the image being executed within each process.

/PROCESS_NAME=process_name
Displays only processes with the specified process name. You can use wildcards in process_name, in which case SDA displays all matching processes. The default action is for SDA to display data for all processes, regardless of process name.

/THREAD
Displays information on all the current threads associated with the current process.

/USER=username
Displays only the processes of the specified user. You can use wildcards in username, in which case SDA displays processes of all matching users. The default action is for SDA to display data for all processes, regardless of username.

Description

The SHOW SUMMARY command displays the information in Table 4–25 for each active process in the system.

Table 4–25 Process Information in the SHOW SUMMARY Display

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended PID</td>
<td>The 32-bit number that uniquely identifies the process.</td>
</tr>
<tr>
<td>Indx</td>
<td>Index of this process into the PCB array.</td>
</tr>
<tr>
<td>Process name</td>
<td>Name assigned to the process.</td>
</tr>
<tr>
<td>Username</td>
<td>Name of the user who created the process.</td>
</tr>
<tr>
<td>State</td>
<td>Current state of the process. Table 4–26 shows the 14 states and their meanings.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 4–25 (Cont.)  Process Information in the SHOW SUMMARY Display

<table>
<thead>
<tr>
<th>Column</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pri</td>
<td>Current scheduling priority of the process.</td>
</tr>
<tr>
<td>PCB/KTB</td>
<td>Address of the process control block or address of the kernel thread block.</td>
</tr>
<tr>
<td>PHD/FRED</td>
<td>Address of the process header or address of the floating-point registers and execution data block.</td>
</tr>
<tr>
<td>Wkset</td>
<td>Number (in decimal) of pages currently in the process working set.</td>
</tr>
</tbody>
</table>

Table 4–26  Current State Information

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM</td>
<td>Computable and resident in memory</td>
</tr>
<tr>
<td>COMO</td>
<td>Computable, but outswapped</td>
</tr>
<tr>
<td>CUR nn</td>
<td>Currently executing on CPU ID nn</td>
</tr>
<tr>
<td>CEF</td>
<td>Waiting for a common event flag</td>
</tr>
<tr>
<td>LEF</td>
<td>Waiting for a local event flag</td>
</tr>
<tr>
<td>LEFO</td>
<td>Outswapped and waiting for a local event flag</td>
</tr>
<tr>
<td>HIB</td>
<td>Hibernating</td>
</tr>
<tr>
<td>HIBO</td>
<td>Hibernating and outswapped</td>
</tr>
<tr>
<td>SUSP</td>
<td>Suspended</td>
</tr>
<tr>
<td>SUSPO</td>
<td>Suspended and outswapped</td>
</tr>
<tr>
<td>PFW</td>
<td>Waiting for a page that is not in memory (page-fault wait)</td>
</tr>
<tr>
<td>FPG</td>
<td>Waiting to add a page to its working set (free-page wait)</td>
</tr>
<tr>
<td>COLPG</td>
<td>Waiting for a page collision to be resolved (collided-page wait); this usually occurs when several processes cause page faults on the same shared page</td>
</tr>
<tr>
<td>MWAIT</td>
<td>Miscellaneous wait</td>
</tr>
<tr>
<td>RWxxx</td>
<td>Waiting for system resource xxx</td>
</tr>
</tbody>
</table>
## SDA Commands

### SHOW SUMMARY

#### Example

```
SDA> SHOW SUMMARY
Current process summary
-----------------------------
<table>
<thead>
<tr>
<th>Extended</th>
<th>Indx</th>
<th>Process name</th>
<th>Username</th>
<th>State</th>
<th>Pri</th>
<th>PCB/KTB</th>
<th>PHD/FRED</th>
<th>Wkset</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000041</td>
<td>0001</td>
<td>SWAPPER</td>
<td>HIB</td>
<td>16</td>
<td>80C641D0 80C63E00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000045</td>
<td>0005</td>
<td>IPCACP</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>10  80D0C0780 81266000</td>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000046</td>
<td>0006</td>
<td>ERRFMT</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>8   80DC2240 8126C000</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000047</td>
<td>0007</td>
<td>OPCOM</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>8   80DC3340 81272000</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000048</td>
<td>0008</td>
<td>AUDIT_SERVER</td>
<td>AUDIT$SERVER</td>
<td>HIB</td>
<td>10  80D61280 81278000</td>
<td>152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000049</td>
<td>0009</td>
<td>JOB_CONTROL</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>10  80D620C0 8127E000</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000004A</td>
<td>000A</td>
<td>SECURITY_SERVER</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>10  80DC58C0 81284000</td>
<td>253</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000004B</td>
<td>000B</td>
<td>TP_SERVER</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>10  80DC8900 8128A000</td>
<td>75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000004C</td>
<td>000C</td>
<td>NETACP</td>
<td>DECNET</td>
<td>HIB</td>
<td>10  80DBFE00 8125A000</td>
<td>78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000004D</td>
<td>000D</td>
<td>EVL</td>
<td>DECNET</td>
<td>HIB</td>
<td>6   80DCA080 81290000</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000004E</td>
<td>000E</td>
<td>REMACP</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>8   80DE4E00 81296000</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000050</td>
<td>0010</td>
<td>DEC$SERVER_0</td>
<td>SYSTEM</td>
<td>HIB</td>
<td>8   80DEF940 812A2000</td>
<td>739</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000051</td>
<td>0011</td>
<td>DEC$LOGINOUT</td>
<td>&lt;login&gt;</td>
<td>LEF</td>
<td>4   80DF0F00 812A8000</td>
<td>273</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000052</td>
<td>0012</td>
<td>SYSTEM</td>
<td>SYSTEM</td>
<td>LEF</td>
<td>9   80D7720C 81260000</td>
<td>75</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The **SHOW SUMMARY** command describes all active processes in the system at the time of the system failure. Note that there was no process in the CUR state at the time of the failure.
SHOW SYMBOL

Displays the hexadecimal value of a symbol and, if the value is equal to an address location, the contents of that location.

Format

SHOW SYMBOL  [/ALL [/ALPHA | /VALUE]] symbol-name

Parameter

symbol-name
Name of the symbol to be displayed. You must provide a symbol-name, unless the /ALL qualifier is specified.

Qualifiers

/ALL
Displays information on all symbols whose names begin with the characters specified in symbol-name. If no symbol name is given, all symbols are displayed.

/ALPHA
When used with the /ALL qualifier, displays the symbols sorted only in alphabetical order. The default is to display the symbols twice, sorted alphabetically and then by value.

When used with a wildcard symbol name, displays the symbols in alphabetical order. This is the default action.

/VALUE
When used with the /ALL qualifier, displays the symbols sorted only in value order. The default is to display the symbols twice, sorted alphabetically and then by value.

When used with a wildcard symbol name, displays the symbols in value order.

Description

The SHOW SYMBOL command with the /ALL qualifier outputs all symbols whose names begin with the characters specified in symbol-name in both alphabetical order and in value order. If no symbol-name is given, all symbols are output.

The SHOW SYMBOL/ALL command is useful for determining the values of symbols that belong to a symbol set, as illustrated in the second example below.

The SHOW SYMBOL command without the /ALL qualifier allows for standard wildcards in the symbol-name parameter. By default, matching symbols are displayed only in alphabetical order. If you specify SHOW SYMBOL/VALUE, then matching symbols are output sorted by value. If you specify SHOW SYMBOL/ALPHA/VALUE, then matching symbols are displayed twice, sorted alphabetically and then by value.

The SHOW SYMBOL command without the /ALL qualifier and no wildcards in the symbol-name parameter outputs the value associated with the given symbol.
When displaying any symbol value, SDA also treats the value as an address and attempts to obtain the contents of the location. If successful, the contents are also displayed.

Examples

1. SDA> SHOW SYMBOL G
   G = FFFFFFFF.80000000 : 6BFA8001.201F0104

   The SHOW SYMBOL command evaluates the symbol G as FFFFFFFF.8000000016 and displays the contents of address FFFFFFFF.8000000016 as 6BFA8001.201F010416.

2. SDA> SHOW SYMBOL/ALL BUG
   Symbols sorted by name----------------------
   BUG$L_BUGCHK_FLAGS = FFFFFFFF.804031E8 : 00000000.00000001
   BUG$L_FATAL_SPSAV = FFFFFFFF.804031F0 : 00000000.00000001
   BUG$REBOOT = FFFFFFFF.8042E320 : 00000000.00001808
   BUG$REBOOT_C = FFFFFFFF.8004F4D0 : 47FB041D.47FD0600

   Symbols sorted by value
   ----------------------
   BUG$REBOOT_C = FFFFFFFF.8004F4D0 :47FB041D.47FD0600
   BUG$L_BUGCHK_FLAGS = FFFFFFFF.804031E8 :00000000.00000001
   BUG$L_FATAL_SPSAV = FFFFFFFF.804031F0 :00000000.00000001
   BUG$REBOOT = FFFFFFFF.8042E320 :00000000.00001808

   This example shows the display produced by the SHOW SYMBOL/ALL command. SDA searches its symbol table for all symbols that begin with the string “BUG” and displays the symbols and their values. Although certain values equate to memory addresses, it is doubtful that the contents of those addresses are actually relevant to the symbol definitions in this instance.
SHOW TQE

Displays the entries in the timer queue. The default output is a summary display of all timer queue entries (TQEs) in chronological order.

Format

SHOW TQE [/ADDRESS=n][/ALL][/BACKLINK][/PID=n]
[/ROUTINE=n]

Parameters

None.

Qualifiers

/ADDRESS=n
Outputs a detailed display of the TQE at the specified address.

/ALL
Outputs a detailed display of all TQEs.

/BACKLINK
Outputs the display of TQEs, either detailed (/ALL) or brief (default), in reverse order, starting at the entry furthest into the future.

/PID=n
Limits the display to the TQEs that affect the process with the specified internal PID. Note that the PID format required is the entire internal PID, including both the process index and the sequence number, and not the extended PID, or process index alone, as used elsewhere in SDA.

/ROUTINE=n
Limits the display to the TQEs for which the specified address is the fork PC.

Description

The SHOW TQE command allows the timer queue to be displayed. By default a summary display of all TQEs is output in chronological order, beginning with the next entry to become current.

The /ADDRESS, /PID, and /ROUTINE qualifiers are mutually exclusive. The /ADDRESS and /BACKLINK qualifiers are mutually exclusive.
In the summary display, the TQE type is given as a five-character code, as in the following:

<table>
<thead>
<tr>
<th>Column</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>Timer ($SETIMR) entry</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>System subroutine entry</td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>Scheduled wakeup ($SCHDWK) entry</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>Single-shot entry</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>Repeated entry</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>Delta time</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>Absolute time</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>CPU time</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>Extended format (64-bit TQE)</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>32-bit TQE</td>
</tr>
</tbody>
</table>

Examples

1. SDA> SHOW TQE

Timer queue entries

System time: 30-MAY-2000 14:49:46.17
First TQE time: 30-MAY-2000 14:49:46.19

<table>
<thead>
<tr>
<th>TQE address</th>
<th>Expiration Time</th>
<th>Type routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>81214040</td>
<td>30-MAY-2000 14:49:46.19</td>
<td>TSD-- 0001000E</td>
</tr>
<tr>
<td>81352780</td>
<td>30-MAY-2000 14:49:46.23</td>
<td>SRD-- 83955BA0</td>
</tr>
<tr>
<td>8126C858</td>
<td>30-MAY-2000 14:49:46.28</td>
<td>SRD-- 81184230</td>
</tr>
<tr>
<td>81211F00</td>
<td>30-MAY-2000 14:49:46.34</td>
<td>SRD-- 8252EAF8</td>
</tr>
<tr>
<td>8103FB28</td>
<td>30-MAY-2000 14:49:46.51</td>
<td>SRD-- 81041930</td>
</tr>
<tr>
<td>81211FC9</td>
<td>30-MAY-2000 14:49:46.56</td>
<td>TSD-- 0001000E</td>
</tr>
<tr>
<td>83975948</td>
<td>30-MAY-2000 14:49:46.79</td>
<td>SRD-- 83974B10</td>
</tr>
<tr>
<td>8131F5C0</td>
<td>30-MAY-2000 14:49:47.00</td>
<td>SRD-- 811FDCD0</td>
</tr>
</tbody>
</table>

This example shows the summary display of all TQEs.

2. SDA> SHOW TQE/ADDRESS=8131F5C0

Timer queue entry 8131F5C0

TQE Address: 8131F5C0 Type: 00000005 SYSTEM_SUBROUTINE REPEAT
FLink: 8129C6D8 BLink: 83975948
Requestor process ID: 00000000 Access Mode: 00000000
Expiration time: 009EADD2.417463F4 30-MAY-2000 15:14:47.31 +67860
Delta repeat time: 00000000.00989680 0 00:00:01.00
Fork PC: 811FDCD0 NETDRIVER+19000
Fork R3: FFFFFFFF.8131DB00

This example shows the detailed display for a single TQE.
SHOW WORKING_SET_LIST, SHOW WSL

Displays the system working set list and retains the current process context.

Format

SHOW WORKING_SET_LIST or SHOW WSL  [=GPT | SYSTEM | LOCKED | n]

Parameters

None.

Qualifiers

None.

Description

The SHOW WORKING_SET_LIST command displays the contents of requested entries in the system working set list. If you do not specify an option, all working set list entries are displayed. Table 4–27 shows the options available with SHOW WORKING_SET_LIST. The SHOW WORKING_SET_LIST command is equivalent to the SHOW PROCESS/SYSTEM/WORKING_SET_LIST command. See the SHOW PROCESS command and Table 4–17 for more information.

<table>
<thead>
<tr>
<th>Options</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT</td>
<td>Displays only working set list entries that are for global page table pages</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Displays only working set list entries for pageable system pages</td>
</tr>
<tr>
<td>LOCKED</td>
<td>Displays only working set list entries for pageable system pages that are locked in the system working set</td>
</tr>
<tr>
<td>n</td>
<td>Displays a specific working set entry, where n is the working set list index (WSLX) of the entry of interest</td>
</tr>
</tbody>
</table>
SPAWN

Creates a subprocess of the process currently running SDA, copying the context of the current process to the subprocess and, optionally, executing a specified command within the subprocess.

Format

SPAWN [/qualifier[...]] [command]

Parameter

command
Name of the command that you want the subprocess to execute.

Qualifiers

/INPUT=filespec
Specifies an input file containing one or more command strings to be executed by the spawned subprocess. If you specify a command string with an input file, the command string is processed before the commands in the input file. Once processing is complete, the subprocess is terminated.

/NOLOGICAL_NAMES
Specifies that the logical names of the parent process are not to be copied to the subprocess. The default behavior is that the logical names of the parent process are copied to the subprocess.

/NOSYMBOLS
Specifies that the DCL global and local symbols of the parent process are not to be passed to the subprocess. The default behavior is that these symbols are passed to the subprocess.

/NOTIFY
Specifies that a message is to be broadcast to SYS$OUTPUT when the subprocess either completes processing or aborts. The default behavior is that such a message is not sent to SYS$OUTPUT.

/NOWAIT
Specifies that the system is not to wait until the subprocess is completed before allowing more commands to be specified. This qualifier allows you to specify new commands while the spawned subprocess is running. If you specify /NOWAIT, use /OUTPUT to direct the output of the subprocess to a file to prevent more than one process from simultaneously using your terminal.

The default behavior is that the system waits until the subprocess is completed before allowing more commands to be specified.

/OUTPUT=filespec
Specifies an output file to which the results of the SPAWN operation are written. To prevent output from the spawned subprocess from being displayed while you are specifying new commands, specify an output other than SYS$OUTPUT whenever you specify /NOWAIT. If you omit the /OUTPUT qualifier, output is written to the current SYS$OUTPUT device.
//PROCESS=process-name
Specifies the name of the subprocess to be created. The default name of the subprocess is USERNAME_n, where USERNAME is the user name of the parent process. The variable n represents the subprocess number.

Example

SDA> SPAWN
$ MAIL
.
.
.
$ DIR
.
.
.
$ LO
  Process SYSTEM_1 logged out at 5-JAN-1993 15:42:23.59
SDA>

This example uses the SPAWN command to create a subprocess that issues DCL commands to invoke the Mail utility. The subprocess then lists the contents of a directory before logging out to return to the parent process executing SDA.
The UNDEFINE command causes SDA to remove the specified symbol from its symbol table.

**Format**

```
UNDEFINE symbol
```

**Parameter**

- `symbol`
  The name of the symbol to be deleted from SDA’s symbol table. A symbol name is required.

**Qualifiers**

- None.
VALIDATE PFN_LIST

Validates that the page counts on lists are correct.

Format

VALIDATE PFN_LIST {/ALL (d)|[/BAD)|[/FREE|[/MODIFIED|[/PRIVATE|[/UNTESTED|[/ZERO]}

Parameters

None.

Qualifiers

/ALL
Validates all the PFN lists: bad, free, modified, zeroed free pages and private pages.

/BAD
Validates the bad page list.

/FREE
Validates the free page list.

/MODIFIED
Validates the modified page list.

/PRIVATE
Validates all private page lists.

/UNTESTED
Validates the untested page list that was set up for deferred memory testing.

/ ZERO
Validates the zeroed free page list.

Description

The VALIDATE PFN_LIST command validates the specified PFN list(s) by counting the number of entries in the list and comparing that to the running count of entries for each list maintained by the system.

Examples

1.  SDA> VALIDATE PFN_LIST
    Free page list validated: 1433 pages
    (excluding zeroed free page list with expected size 103 pages)
    Zeroed free page list validated: 103 pages
    Modified page list validated: 55 pages
    Bad page list validated: 0 pages
    Untested page list validated: 0 pages
    Private page list at 81486340 validated: 2 pages
2. SDA> VALIDATE PFN_LIST/FREE
   Free page list validated: 1433 pages
   (excluding zeroed free page list with expected size 103 pages)
VALIDATE QUEUE

Validates the integrity of the specified queue by checking the pointers in the queue.

Format

VALIDATE QUEUE  [address]
               [/BACKLINK | /LIST | /PHYSICAL |
               /QUADWORD | /SELF_RELATIVE | /SINGLY_LINKED]

Parameter

address
Address of an element in a queue.
If you specify the period (. ) as the address, SDA uses the last evaluated expression as the queue element's address.
If you do not specify an address, the VALIDATE QUEUE command determines the address from the last issued VALIDATE QUEUE command in the current SDA session.
If you do not specify an address, and no queue has previously been specified, SDA displays the following error message:
%SDA-E-NOQUEUE, no queue has been specified for validation

Qualifiers

/BACKLINK
Allows doubly linked lists to be validated from the tail of the queue. If the queue is found to be broken when validated from the head of the queue, you can use /BACKLINK to narrow the list of corrupted entries.

/LIST
Displays the address of each element in the queue.

/PHYSICAL
Allows validation of queues whose header and links are physical addresses.

/QUADWORD
Allows the validate operation to occur on queues with linked lists of quadword addresses.

/SELF_RELATIVE
Specifies that the selected queue is a self-relative queue.

/SINGLY_LINKED
Allows validation of queues that have no backward pointers.
SDA Commands
VALIDATE QUEUE

Description
The VALIDATE QUEUE command uses the forward, and optionally, backward pointers in each element of the queue to make sure that all such pointers are valid and that the integrity of the queue is intact. If the queue is intact, SDA displays the following message:

Queue is complete, total of n elements in the queue

In these messages, n represents the number of entries the VALIDATE QUEUE command has found in the queue.

If SDA discovers an error in the queue, it displays one of the following error messages:

Error in forward queue linkage at address nnnnnnnn after tracing x elements
Error comparing backward link to previous structure address (nnnnnnnn)
Error occurred in queue element at address oooooooo after tracing pppp elements

These messages can appear frequently when you use the VALIDATE QUEUE command within an SDA session that is analyzing a running system. In a running system, the composition of a queue can change while the command is tracing its links, thus producing an error message.

If there are no entries in the queue, SDA displays this message:

The queue is empty

Examples

1. SDA> VALIDATE QUEUE/SELF_RELATIVE IOC$GQ_POSTIQ
Queue is complete, total of 159 elements in the queue

This example validates the self-relative queue IOC$GQ_POSTIQ. The validation is successful and the system determines that there are 159 IRPs in the list.

2. SDA> VALIDATE QUEUE/QUADWORD FFFFFFFF80D0E6CO/LIST

<table>
<thead>
<tr>
<th>Entry</th>
<th>Address</th>
<th>Flink</th>
<th>Blink</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>FFFFFFFF80D0E6CO</td>
<td>FFFFFFFF80D03780</td>
<td>FFFFFFFF80D0E800</td>
</tr>
<tr>
<td>1.</td>
<td>FFFFFFFF80D0E790</td>
<td>FFFFFFFF80D0E7C0</td>
<td>FFFFFFFF80D0E6C0</td>
</tr>
<tr>
<td>2.</td>
<td>FFFFFFFF80D0E800</td>
<td>FFFFFFFF80D0E6C0</td>
<td>FFFFFFFF80D0E7C0</td>
</tr>
</tbody>
</table>

Queue is complete, total of 3 elements in the queue

This example shows the validation of quadword elements in a list.

3. SDA> VALIDATE QUEUE/SINGLY_LINKED EXE$GL_NONPAGED+4
Queue is zero-terminated, total of 95 elements in the queue

This example shows the validation of singly linked elements in the queue. The forward link of the final element is zero instead of being a pointer back to the queue header.
VALIDATE SHM_CPP

Validates all the shared memory common property partitions (CPPs) and the counts and ranges of attached PFNs; optionally, it can validate the contents of the database for each PFN.

Format

VALIDATE SHM_CPP [/QUALIFIERS]

Parameters

None.

Qualifiers

/ADDRESS=n
Validates the counts and ranges for a single shared memory CPP given the address of the SHM_CPP structure.

/ALL
Validates all the shared memory CPPs. This is the default.

/IDENT=n
Validates the counts and ranges for a single shared memory CPP.

/PFN
Validates the PFN database contents for each attached PFN. The default is all lists (free, bad, untested) plus the PFN database pages and the complete range of PFNs in the CPP.

To validate only the complete range of PFNs in the CPP, use the keyword ALL_FRAGMENTS with the /PFN qualifier:

/PFN = ALL_FRAGMENTS

To validate only the bad page list, use the keyword BAD with the /PFN qualifier:

/PFN = BAD

To validate only the free page list, use the keyword FREE with the /PFN qualifier:

/PFN = FREE

To validate the PFNs containing the PFN database, use the keyword PFNDB with the /PFN qualifier:

/PFN = PFNDB

To validate only the untested page list, use the keyword UNTESTED with the /PFN qualifier:

/PFN = UNTESTED

To validate multiple lists, you can combine keywords for use with the /PFN qualifier:

/PFN = (x,y)

Note that if the /PFN is given without /ALL, /IDENT, or /ADDRESS, then the system validates the PFN lists from the last shared memory CPP.
Example

SDA> SHOW SHM_CPP
Not validating SHM_CPP 0000 at FFFFFFFF.7F2BA140, VALID flag clear
Not validating SHM_CPP 0001 at FFFFFFFF.7F2BA380, VALID flag clear
Not validating SHM_CPP 0002 at FFFFFFFF.7F2BA5C0, VALID flag clear
Validating SHM_CPP 0003 at FFFFFFFF.7F2BA800 ...
   Validating counts and ranges in the free page list ...
   ... o.k.
   Not validating the bad page list, list is empty
   Not validating the untested page list, list is empty
Not validating SHM_CPP 0004 at FFFFFFFF.7F2BAAC0, VALID flag clear
Not validating SHM_CPP 0005 at FFFFFFFF.7F2BAC80, VALID flag clear
Not validating SHM_CPP 0006 at FFFFFFFF.7F2BAEC0, VALID flag clear

This example shows the default output for the VALIDATE SHM_CPP command.
This chapter presents an overview of the SDA CLUE (Crash Log Utility Extractor) extension commands, how to display information using these commands, and how to use SDA CLUE with DOSD. This chapter also describes the SDA CLUE commands.

5.1 Overview of SDA CLUE Extensions

SDA CLUE (Crash Log Utility Extractor) commands automate the analysis of crash dumps and maintain a history of all fatal bugchecks on either a standalone or cluster system. You can use SDA CLUE commands in conjunction with SDA to collect and decode additional dump file information not readily accessible through standard SDA commands. SDA CLUE extension commands can summarize information provided by certain standard SDA commands and provide additional detail for some SDA commands. For example, SDA CLUE extension commands can quickly provide detailed extended QIO processor (XOP) summaries. You can also use SDA CLUE commands interactively on a running system to help identify performance problems.

You can use all CLUE commands when analyzing crash dumps; the only CLUE commands that are not allowed when analyzing a running system are CLUE CRASH, CLUE ERRLOG, CLUE HISTORY, and CLUE STACK.

When you reboot the system after a system failure, you automatically invoke SDA by default. To facilitate better crash dump analysis, SDA CLUE commands automatically capture and archive summary dump file information in a CLUE listing file.

A startup command procedure initiates commands that do the following:

- Invoke SDA
- Issue an SDA CLUE HISTORY command
- Create a listing file called CLUE$nodename_ddmmyy_hhmm.LIS

The CLUE HISTORY command adds a one-line summary entry to a history file and saves the following output from SDA CLUE commands in the listing file:

- Crash dump summary information
- System configuration
- Stack decoder
- Page and swap files
5.1 Overview of SDA CLUE Extensions

- Memory management statistics
- Process DCL recall buffer
- Active XQP processes
- XQP cache header

The contents of this CLUE list file can help you analyze a system failure. If these files accumulate more space than the threshold allows (default is 5000 blocks), the oldest files are deleted until the threshold limit is reached. You can also customize this list file using the CLUE$MAX_BLOCK logical name.

For additional information on the contents of the CLUE listing file, see the reference section on CLUE HISTORY.

It is important to remember that CLUE$nodename_ddmmyy_hhmm.LIS contains only an overview of the crash dump and does not always contain enough information to determine the cause of the crash. The dump itself should always be saved using the procedures described in Section 2.2.2 and Section 2.2.3.

To inhibit the running of CLUE at system startup, define the logical CLUE$INHIBIT in the SYLOGICALS.COM file as /SYS TRUE.

5.2 Displaying Data Using SDA CLUE Commands

To invoke a CLUE command, enter the command at the SDA prompt. For example:

SDA> CLUE CONFIG

5.3 Using SDA CLUE with DOSD

DOSD (Dump Off System Disk) allows you to write the system dump file to a device other than the system disk. For SDA CLUE to be able to correctly find the dump file to be analyzed after a system crash, you need to perform the following steps:

1. Modify the command procedure SYS$MANAGER:SYCONFIG.COM to add the system logical name CLUE$DOSD_DEVICE to point to the device where the dump file resides. You need to supply only the physical or logical device name without a file specification.

2. Modify the command procedure SYS$MANAGER:SYCONFIG.COM to mount systemwide the device where the dump file resides. Otherwise, SDA CLUE cannot access and analyze the dump file.

In the following example, the dump file has been placed on device $3$DUA25, which has the label DMP$DEV. You need to add the following commands to SYS$MANAGER:SYCONFIG.COM:

$mount/system/noassist $3$dua25: dmp$dev dmp$dev
$define/system clue$dosd_device dmp$dev
5.4 Listing of SDA CLUE Extension Commands

This section describes the following SDA CLUE extension commands:

- CLUE CALL_FRAME
- CLUE CLEANUP
- CLUE CONFIG
- CLUE CRASH
- CLUE ERRLOG
- CLUE FRU
- CLUE HISTORY
- CLUE MCHK
- CLUE MEMORY
- CLUE PROCESS
- CLUE REGISTER
- CLUE SG
- CLUE STACK
- CLUE SYSTEM
- CLUE VCC
- CLUE XQP
SDA CLUE Extension Commands

CLUE CALL_FRAME

Displays key information, such as the PC of the caller, from the active call frames at time of the crash.

Format

```
CLUE CALL_FRAME [/CPU [cpu-id | ALL]
 | /PROCESS [/ADDRESS=n | INDEX=n
 | /IDENTIFICATION=n | process-name | ALL]]
```

Parameters

**ALL**
When used with /CPU, it requests information about all CPUs in the system. When used with /PROCESS, it requests information about all processes that exist in the system.

**cpu-id**
When used with /CPU, it gives the number of the CPU for which information is to be displayed. Use of the *cpu-id* parameter causes the CLUE CALL_FRAME command to perform an implicit SET CPU command, making the indicated CPU the current CPU for subsequent SDA commands.

**process-name**
When used with /PROCESS, it gives the name of the process for which information is to be displayed. Use of the *process-name* parameter, the /ADDRESS qualifier, the /INDEX qualifier, or the /IDENTIFICATION qualifier causes the CLUE CALL_FRAME command to perform an implicit SET PROCESS command, making the indicated process the current process for subsequent SDA commands. You can determine the names of the processes in the system by issuing a SHOW SUMMARY command.

The *process-name* can contain up to 15 letters and numerals, including the underscore (_ ) and dollar sign ($ ). If it contains any other characters, you must enclose the *process-name* in quotation marks (" ").

Qualifiers

**/ADDRESS=**
Specifies the PCB address of the desired process when used with CLUE CALL_FRAME/PROCESS.

**/CPU **[cpu-id | ALL]
Indicates that the call frame for a CPU is required. Specify the CPU by its number or use ALL to indicate all CPUs.

**/IDENTIFICATION=**
Specifies the identification of the desired process when used with CLUE CALL_FRAME/PROCESS.

**/INDEX=**
Specifies the index of the desired process when used with CLUE CALL_FRAME/PROCESS.
/PROCESS [process-name | ALL]
Indicates that the call frame for a process is required. The process should be specified with either one of the qualifiers /ADDRESS, /IDENTIFICATION, or /INDEX, or by its name, or by using ALL to indicate all processes.

Description
The CLUE CALL_FRAME command displays call chain information for a process or a CPU. The process context calls work on both the running system and dump file; the CPU context calls only on dump files.

If neither /CPU nor /PROCESS is specified, the parameter (CPU-id or process-name) is ignored and the call frame for the SDA current process is displayed.

Examples
1. SDA> CLUE CALL/PROCESS IPCACP
   Call Chain: Process index: 000B Process name: IPCACP PCB: 8136EF00
   Procedure Frame Procedure Entry Return Address
   -------------- ------------------ -----------------------------
   7FFA1CA0 Null 800C8C90 SCH$WAIT_PROC_C
   7FFA1D00 Stack 800D9250 SYS$HIBER_C 0003045C IPCACP+0003045C
   7FFA1D50 Stack 00030050 IPCACP+00030050 800D11C8 EXE$CMKRNL_C+000D8
   7FFA1E60 Null 800B6120 EXE$BLDPKTSWPR_C
   7FFA1E78 Null 800B6120 EXE$BLDPKTSWPR_C
   7FFA1ECC Null 80248120 NSA$CHECK_PRIVILEGE_C
   7FFA1FC0 Null 80084640 EXE$CMODEEXECX_C
   7FFA1FC70 Stack 800D10F0 EXE$CMKRNL_C 80084CC8 EXE$CMODKRNL_C+00198
   7FB01FAB0 Stack 00030010 IPCACP+00030010 83EA3454 SYS$IMGSTA_C+00154
   7FB01FB10 Stack 83EA3300 SYS$IMGSTA_C 83D99CC4 EXE$PROC_IMGACT_C+00384
   7FB01FB20 Stack 83D99BA0 EXE$PROC_IMGACT_C+00260 83D99B9C EXE$PROC_IMGACT_C+0025C

   In this example, the CLUE CALL_FRAME command displays the call frame from the process IPCACP.

2. SDA> CLUE CALL/CPU ALL
   Call Chain: Process index: 0000 Process name: NULL PCB: 827377C0 (CPU 0)
   Procedure Frame Procedure Entry Return Address
   -------------- ------------------ -----------------------------
   8F629D28 Null 80205E00 SYS$SCS+05E00
   8F629D68 Null 8020A850 SC$REC_MSGREC_C
   8F629D98 Null 914A5340 SYS$PBDRIVER+07340
   8F629DB8 Null 914A4FD0 SYS$PBDRIVER+06FD0
   8F629DDE Stack 914AA4CF0 SYS$PBDRIVER+0CCF0 914AE5CC SYS$PBDRIVER+105CC
   8F629E50 Stack 914AE418 SYS$PBDRIVER+10418 800503B0 EXE_STD$QUEUE_FORK_C+00350
   8F629FF8 Null 800E95F4 SCH$WAIT_ANY_MODE_C
   8F629FFD Stack 800D0F80 SCH$IDLE_C 800E92D0 SCH$INTERRUPT+00BB0

   Call Chain: Process index: 0000 Process name: NULL PCB: 827377C0 (CPU 2)
   Procedure Frame Procedure Entry Return Address
   -------------- ------------------ -----------------------------
   90FCBF88 Null 800E95F4 SCH$WAIT_ANY_MODE_C
   90FCBF8C Null 800E95F4 SCH$WAIT_ANY_MODE_C
   90FCBFFD Stack 800D0F80 SCH$IDLE_C 800E92D0 SCH$INTERRUPT+00BB0
### SDA CLUE Extension Commands

**CLUE CALL_FRAME**

<table>
<thead>
<tr>
<th>Procedure Frame</th>
<th>Procedure Entry</th>
<th>Return Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>90FCBF88</td>
<td>Null</td>
<td>800E95FA</td>
</tr>
<tr>
<td>90FD9F88</td>
<td>Null</td>
<td>800E95F4</td>
</tr>
<tr>
<td>90FD9FD0</td>
<td>Stack</td>
<td>800D0F80</td>
</tr>
</tbody>
</table>

In this example, CLUE/CPU ALL shows the call frame for all CPUs.
CLUE CLEANUP

Performs housekeeping operations to conserve disk space.

Format

CLUE CLEANUP

Parameters

None.

Qualifiers

None.

Description

CLUE CLEANUP performs housekeeping operations to conserve disk space. To avoid filling up the system disk with listing files generated by CLUE, CLUE CLEANUP is run during system startup to check the overall disk space used by all CLUE$*.LIS files.

If the CLUE$COLLECT:CLUE$*.LIS files occupy more space than the logical CLUE$MAX_BLOCKS allows, then the oldest files are deleted until the threshold is reached. If this logical name is not defined, a default value of 5,000 disk blocks is assumed. A value of zero disables housekeeping and no check on the disk space is performed.

Example

SDA> CLUE CLEANUP
%CLUE-I-CLEANUP, housekeeping started...
%CLUE-I-MAXBLOCK, maximum blocks allowed 5000 blocks
%CLUE-I-STAT, total of 4 CLUE files, 192 blocks.

In this example, the CLUE CLEANUP command displays that the total number of blocks of disk space used by CLUE files does not exceed the maximum number of blocks allowed. No files are deleted.
SDA CLUE Extension Commands

CLUE CONFIG

CLUE CONFIG

Displays the system, memory, and device configurations.

Format

CLUE CONFIG

Parameters

None.

Qualifiers

None.

Description

CLUE CONFIG displays the system, memory, and device configurations.

Example

SDA> CLUE CONFIG
System Configuration:
System Information:
System Type AlphaServer 4100 5/400 4MB Primary CPU ID 00
Cycle Time 2.5 nsec (400 MHz) Pagesize 8192 Byte
Memory Configuration:
Cluster PPN Start PPN Count Range (MByte) Usage
#00 0 256 0.0 MB - 2.0 MB Console
#01 256 32510 2.0 MB - 255.9 MB System
Per-CPU Slot Processor Information:
CPU ID 00 CPU State rr,pa,pp,cv,pv,pmv,pl
PAL Code 1.19-12 Halt Request "Default, No Action"
CPU Revision .... Halt PS 00000000.00000000
Serial Number ........ Halt Code "Bootstrap or Powerfail"
Console Vers V5.0-47
CPU ID 02 CPU State pa,pp,cv,pv,pmv,pl
CPU Type EV56 Pass 2 (21164A) Halt Request "Default, No Action"
PAL Code 1.19-12 Halt PC 00000000.00000000
CPU Revision .... Halt PS 00000000.00000000
Serial Number ........ Halt Code "Bootstrap or Powerfail"
Console Vers V5.0-47
Adapter Configuration:

VM-0011A-AI

5–8 SDA CLUE Extension Commands
CLUE CRASH

Displays a crash dump summary.

Format

CLUE CRASH

Parameters

None.

Qualifiers

None.

Description

CLUE CRASH displays a crash dump summary, which includes the following items:

- Bugcheck type
- Current process and image
- Failing PC and PS
- Executive image section name and offset
- General registers
- Failing instructions
- Exception frame, signal and mechanism arrays (if available)
- CPU state information (spinlock related bugchecks only)

Example

SDA> CLUE CRASH
Crash Time: 30-AUG-1996 13:13:46.83
Bugcheck Type: SSRRVEXCEPT, Unexpected system service exception
Node: SWPCTX (Standalone)
CPU Type: DEC 3000 Model 400
VMS Version: X6AF-FT2
Current Process: SYSTEM
Current Image: $31$DKB0:[SYS0.][SYSMGR].X.EXE;1
Failing PC: 00000000.00030078 SYS$K_VERSION_01+00078
Failing PS: 00000000.00000003
Module: X
Offset: 00000078
CLUE CRASH

Boot Time: 30-AUG-1996 09:06:22.00
System Uptime: 04:07:24.83
Crash/Primary CPU: 00/00
System/CPU Type: 0402
Saved Processes: 18
Pagesize: 8 KByte (8192 bytes)
Physical Memory: 64 MByte (8192 PFNs, contiguous memory)
Dumpfile Pagelets: 98861 blocks
Dump Flags: olddump,writecomp,errlogcomp,dump_style
Dump Type: raw,selective
EXESG1_FLAGS: poolpging,init,bugdump
Paging Files: 1 Pagefile and 1 Swapfile installed

Stack Pointers:
KSP = 00000000.7FFA1C98 ESP = 00000000.7FFA6000 SSP = 00000000.7FFAC100
USP = 00000000.7AFFBAD0

General Registers:
R0 = 00000000.00000000 R1 = 00000000.7FFA1EB8 R2 = FFFFFFFF.80D0E6C0
R3 = FFFFFF80C63460 R4 = FFFFFF.80D12740 R5 = 00000000.000000C8
R6 = 00000000.00030038 R7 = 00000000.7FFA1FC0 R8 = 00000000.7FFAC208
R9 = 00000000.7FFAC410 R10 = 00000000.7FFAD238 R11 = 00000000.7FFCE23E0
R12 = 00000000.00000000 R13 = FFFFFF80C66EB60 R14 = 00000000.00000000
R15 = 00000000.009A79FD R16 = 00000000.00000000 R17 = 00000000.7FFA1D40
R18 = FFFFFF80C505C38 R19 = 00000000.00000000 R20 = 00000000.7FFA1F50
R21 = 00000000.00000000 R22 = 00000000.00000000 R23 = 00000000.7FF03C8
R24 = 00000000.7FF0040 R25 = 00000000.00000000 R26 = FFFFFFFF.8010ACA4
R27 = 00000000.00000000 R28 = 00000000.7FFF0040 AI = 00000000.00000003
R29 = 00000000.00010051 R30 = 00000000.7FFA1D40 PS = 00000000.00000000

Exception Frame:
R2 = 00000000.00000000 R3 = FFFFFF80C63460 R4 = FFFFFF.80D12740
R5 = 00000000.000000C8 R6 = 00000000.00030038 R7 = 00000000.7FFA1FC0
PC = 00000000.00030078 PS = 00000000.00000003

Signal Array: 64-bit Signal Array:
Arg Count = 00000005 Arg Count = 00000005
Condition = 00000000 Condition = 00000000
Argument #2 = 00000000.00000000 Argument #2 = 00000000.00000000
Argument #3 = 00000000.00000000 Argument #3 = 00000000.00000000
Argument #4 = 00000000.00000000 Argument #4 = 00000000.00000000
Argument #5 = 00000000.00000000 Argument #5 = 00000000.00000000

Mechanism Array:
Arguments = 00000002C Establisher FP = 00000000.7AFFBAD0
Flags = 00000000 Exception FP = 00000000.7FFA1F00
Depth = FFFFFFED Signal Array = 00000000.7FFA1EB8
Handler Data = 00000000.00000000 Signal64 Array = 00000000.7FFA1ED0
R0 = 00000000.00020000 R1 = 00000000.00000000 R16 = 00000000.00020004
R17 = 00000000.00010050 R18 = FFFFFF.FFFFF0 R19 = 00000000.00000000
R20 = 00000000.7FFA1F50 R21 = 00000000.00000000 R22 = 00000000.00001005
R23 = 00000000.00000000 R24 = 00000000.00010051 R25 = 00000000.00000000
R26 = FFFFFF8010AC4 R27 = 00000000.00010050 R28 = 00000000.00000000

System Registers:
Page Table Base Register (PTBR) = 00000000.0001136
Processor Base Register (PRBR) = FFFFFF.80D0E000
Privileged Context Block Base (PCBB) = 00000000.0003F008
System Control Block Base (SCBB) = 00000000.000001DC
Software Interrupt Summary Register (SISR) = 00000000.00000000
Address Space Number (ASN) = 00000000.0000002F
AST Summary / AST Enable (ASTSR_ASTEN) = 00000000.00000000
Floating-Point Enable (FEN) = 00000000.00000000
Interrupt Priority Level (IPL) = 00000000.00000000
Machine Check Error Summary (MCES) = 00000000.00000000
Virtual Page Table Base Register (VPTB) = FFFFFF8000000000
Failing Instruction:
SYS$K_VERSION_01+000078:  LDL R28,(R28)

Instruction Stream (last 20 instructions):
SYS$K_VERSION_01+000028:  LDQ R16,#X0030(R13)
SYS$K_VERSION_01+00002C:  LDQ R27,#X0048(R13)
SYS$K_VERSION_01+000030:  LDA R17,(R28)
SYS$K_VERSION_01+000034:  JSR R26,(R26)
SYS$K_VERSION_01+000038:  LDQ R26,#X0038(R13)
SYS$K_VERSION_01+00003C:  BIS R31,SP,SP
SYS$K_VERSION_01+000040:  BIS R31,R26,RO
SYS$K_VERSION_01+000044:  BIS R31,FP,SP
SYS$K_VERSION_01+000048:  LDQ R28,#X0008(SP)
SYS$K_VERSION_01+00004C:  LDQ R13,#X0010(SP)
SYS$K_VERSION_01+000050:  LDQ FP,#X0018(SP)
SYS$K_VERSION_01+000054:  LDA SP,#X0020(SP)
SYS$K_VERSION_01+000058:  RET R31,(R28)
SYS$K_VERSION_01+00005C:  BIS R31,R31,R31
SYS$K_VERSION_01+000060:  LDA SP,#XFFE0(SP)
SYS$K_VERSION_01+000064:  STQ FP,#X0018(SP)
SYS$K_VERSION_01+000068:  STQ R27,(SP)
SYS$K_VERSION_01+00006C:  BIS R31,SP,FP
SYS$K_VERSION_01+000070:  STQ R26,#X0010(SP)
SYS$K_VERSION_01+000074:  LDA R28,(R31)
SYS$K_VERSION_01+000078:  LDL R28,(R28)
SYS$K_VERSION_01+00007C:  BEQ R26,#X000007
SYS$K_VERSION_01+000080:  LDQ R26,#XFFE8(R27)
SYS$K_VERSION_01+000084:  BIS R31,R26,RO
SYS$K_VERSION_01+000088:  BIS R31,FP,SP
CLUE ERRLOG

Extracts the error log buffers from the dump file and places them into the binary file called CLUE$ERRLOG.SYS.

Format

```
CLUE ERRLOG [/OLD]
```

Parameters

None.

Qualifier

/OLD

Dumps the errorlog buffers into a file using the old errorlog format. The default action, if /OLD is not specified, is to dump the errorlog buffers in the common event header format.

Description

CLUE ERRLOG extracts the error log buffers from the dump file and places them into the binary file called CLUE$ERRLOG.SYS.

These buffers contain messages not yet written to the error log file at the time of the failure. When you analyze a failure on the same system on which it occurred, you can run the Error Log utility on the actual error log file to see these error log messages. When analyzing a failure from another system, use the CLUE ERRLOG command to create a file containing the failing system’s error log messages just prior to the failure. System failures are often triggered by hardware problems, so determining what, if any, hardware errors occurred prior to the failure can help you troubleshoot a failure.

You can define the logical CLUE$ERRLOG to any file specification if you want error log information written to a file other than CLUE$ERRLOG.SYS.

Note

You need at least DECevent V2.9 to analyze the new common event header (CEH) format file. The old format file can be analyzed by ANALYZE/ERROR or any version of DECevent.

Example

```
SDA> CLUE ERRLOG
Sequence Date Time
-------- ----------- -----------
128 11-MAY-1994 00:39:31.30
129 11-MAY-1994 00:39:32.12
130 11-MAY-1994 00:39:44.83
131 11-MAY-1994 00:44:38.97 * Crash Entry
```

In addition to writing the error log buffers into CLUE$ERRLOG.SYS, the CLUE ERRLOG command displays the sequence, date, and time of each error log buffer extracted from the dump file.
CLUE FRU

Outputs the Field Replacement Unit (FRU) table to a file for display by DECevent.

Format

CLUE FRU

Parameters

None.

Qualifiers

None.

Description

The FRU command extracts the FRU table into an output file (CLUE$FRU.SYS), which can then be displayed by DECevent. This command works on the running system, as well as on dump files.
CLUE HISTORY

Updates history file and generates crash dump summary output.

Format

CLUE HISTORY [/qualifier]

Parameters

None.

Qualifier

/OVERRIDE

Allows execution of this command even if the dump file has already been analyzed (DMP$¥_OLDDUMP bit set).

Description

This command updates the history file pointed to by the logical name CLUE$HISTORY with a one-line entry and the major crash dump summary information. If CLUE$HISTORY is not defined, a file CLUE$HISTORY.DAT in your default directory will be created.

In addition, a listing file with summary information about the system failure is created in the directory pointed to by CLUE$COLLECT. The file name is of the form CLUE$node_ddmmyy_hhmm.LIS where the timestamp (hhmm) corresponds to the system failure time and not the time when the file was created.

The listing file contains summary information collected from the following SDA commands:

• CLUE CRASH
• CLUE CONFIG
• CLUE MEMORY/FILES
• CLUE MEMORY/STATISTIC
• CLUE PROCESS/RECALL
• CLUE XQP_ACTIVE

Refer to the reference section for each of these commands to see examples of the displayed information.

The logical name CLUE$FLAG controls how much information is written to the listing file.

• Bit 0—Include crash dump summary
• Bit 1—Include system configuration
• Bit 2—Include stack decoding information
• Bit 3—Include page and swap file usage
• Bit 4—Include memory management statistics
• Bit 5—Include process DCL recall buffer
• Bit 6—Include active XQP process information
• Bit 7—Include XQP cache header

If this logical name is undefined, all bits are set by default internally and all information is written to the listing file. If the value is zero, no listing file is generated. The value has to be supplied in hexadecimal form (for example, DEFINE CLUE$FLAG 81 will include the crash dump summary and the XQP cache header information).

If the logical name CLUESITE_PROC points to a valid and existing file, it will be executed as the final step of the CLUE HISTORY command (for example, automatic saving of the dump file during system startup). If used, this file should contain only valid SDA commands.

Refer to Chapter 2, Section 2.2.3 for more information on site-specific command files.
CLUE MCHK

This command is obsolete.

Format

CLUE MCHK

Parameters

None.

Qualifiers

None.

Description

The CLUE MCMK command has been withdrawn. Issuing the command produces the following output, explaining the correct way to obtain MACHINECHECK information from a crash dump.

Please use the following commands in order to extract the errorlog buffers from the dumpfile header and analyze the machine check entry:

$ analyze/crash sys$system:sysdump.dmp
SDA> clue errlog
SDA> exit
$ diagnose clue$errlog
CLUE MEMORY

Displays memory- and pool-related information.

Format

CLUE MEMORY [/qualifier[,...]]

Parameters

None.

Qualifiers

/FILES
Displays information about page and swap file usage.

/FREE [/FULL]
Validates and displays dynamic nonpaged free packet list queue.

/GH [/FULL]
Displays information about the granularity hint regions.

/LAYOUT
Decodes and displays much of the system virtual address space layout.

/LOOKASIDE
Validates the lookaside list queue heads and counts the elements for each list.

/STATISTIC
Displays systemwide performance data such as page fault, I/O, pool, lock manager, MSCP, and file system cache.

Description

The CLUE MEMORY command displays memory- and pool-related information.

Examples

1. SDA> CLUE MEMORY/FILES

Paging File Usage (blocks):

<table>
<thead>
<tr>
<th></th>
<th>Device</th>
<th>DKA0:</th>
<th>Swapfile</th>
<th>Index  1</th>
<th>PFL Address</th>
<th>8FF61531340</th>
<th>UCB Address</th>
<th>8FF615313E0</th>
<th>Free Blocks</th>
<th>44288</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size (blocks)</td>
<td>44288</td>
<td>Flags</td>
<td>inited, swap_file</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Write Count</td>
<td>0</td>
<td>Total Read Count</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest Chunk (pages)</td>
<td>2768</td>
<td>Largest Chunk (pages)</td>
<td>2768</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chunks GEQ 64 Pages</td>
<td>1</td>
<td>Chunks LT 64 Pages</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Device</th>
<th>DKA0:</th>
<th>Pagefile</th>
<th>Index 254</th>
<th>PFL Address</th>
<th>8FF61532B440</th>
<th>UCB Address</th>
<th>8FF61532B00</th>
<th>Free Blocks</th>
<th>1056768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size (blocks)</td>
<td>1056768</td>
<td>Flags</td>
<td>inited</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Write Count</td>
<td>0</td>
<td>Total Read Count</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest Chunk (pages)</td>
<td>66048</td>
<td>Largest Chunk (pages)</td>
<td>66048</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chunks GEQ 64 Pages</td>
<td>1</td>
<td>Chunks LT 64 Pages</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary: 1 Pagefile and 1 Swapfile installed
Total Size of all Swap Files: 44288 blocks
Total Size of all Paging Files: 1056768 blocks
Total Committed Paging File Usage: 344576 blocks

This example shows the display produced by the CLUE MEMORY/FILES command.

2. SDA> CLUE MEMORY/FREE/FULL
Non-Paged Dynamic Storage Pool - Variable Free Packet Queue:

CLASSDR FFFFFFFF.80D157C0 : 64646464 64646464 00000040 80D164C0 Àdñ.8...ddddd
düñA...dddddddd

VCC FFFFFFFF.80D228C0 : 801CA5E8 026F0040 00000040 80D23E40 @>0.8...@.o.eY...

Largest free chunk: 00031C40 (hex) 203840 (dec) bytes
Total free dynamic space: 0003D740 (hex) 251712 (dec) bytes

The CLUE MEMORY/FREE/FULL command validates and displays dynamic nonpaged free packet list queue.

3. SDA> CLUE MEMORY/GH/FULL
Granularity Hint Regions - Huge Pages:

<table>
<thead>
<tr>
<th>Execlet Code Region</th>
<th>Pages/Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base/End VA</td>
<td>FFFFFFFF.80000000 FFFFFFFF.80356000 Current Size 427/ 427</td>
</tr>
<tr>
<td>Base/End PA</td>
<td>00000000.00400000 00000000.00756000 Free / 0</td>
</tr>
<tr>
<td>Total Size</td>
<td>00000000.00356000 3.3 MB In Use / 427</td>
</tr>
<tr>
<td>Bitmap VA/Size</td>
<td>FFFFFFFF.80D17CC0 00000000.00000040 Initial Size 512/ 512</td>
</tr>
<tr>
<td>Slice Size</td>
<td>00000000.00000200 Released 85/ 85</td>
</tr>
<tr>
<td>Next free Slice</td>
<td>00000000.000001AB</td>
</tr>
</tbody>
</table>

Free Packet Queue, Status: Valid, 174 elements
### SDA CLUE Extension Commands

#### CLUE MEMORY

<table>
<thead>
<tr>
<th>Image Base</th>
<th>End</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS$PUBLIC_VECTORS</td>
<td>FFFFFFF.80000000</td>
<td>FFFFFFF.8001A000</td>
</tr>
<tr>
<td>SYS$BASE_IMAGE</td>
<td>FFFFFFF.80002000</td>
<td>FFFFFFF.8000D400</td>
</tr>
<tr>
<td>SYS$CNBTDriver</td>
<td>FFFFFFF.8000E000</td>
<td>FFFFFFF.8000F000</td>
</tr>
<tr>
<td>SYS$SNISCA_BTDriver</td>
<td>FFFFFFF.80010000</td>
<td>FFFFFFF.8001F800</td>
</tr>
<tr>
<td>SYS$SESBDriver</td>
<td>FFFFFFF.80020000</td>
<td>FFFFFFF.80022400</td>
</tr>
<tr>
<td>SYS$DOPDriver</td>
<td>FFFFFFF.80024000</td>
<td>FFFFFFF.80027C00</td>
</tr>
<tr>
<td>SYSTEM_DEBUG</td>
<td>FFFFFFF.80028000</td>
<td>FFFFFFF.80050200</td>
</tr>
<tr>
<td>SYSTEM_PRIMITIVES</td>
<td>FFFFFFF.80052000</td>
<td>FFFFFFF.80089000</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION</td>
<td>FFFFFFF.8008A000</td>
<td>FFFFFFF.80095400</td>
</tr>
<tr>
<td>ERRORLOG</td>
<td>FFFFFFF.80096000</td>
<td>FFFFFFF.80099200</td>
</tr>
<tr>
<td>SYS$CPU_Routines_0402</td>
<td>FFFFFFF.800A4000</td>
<td>FFFFFFF.800A9000</td>
</tr>
<tr>
<td>EXCEPTION_MON</td>
<td>FFFFFFF.800B8000</td>
<td>FFFFFFF.800BC800</td>
</tr>
<tr>
<td>IO_Routines_MON</td>
<td>FFFFFFF.800BE000</td>
<td>FFFFFFF.800E2000</td>
</tr>
<tr>
<td>SYSDEVICE</td>
<td>FFFFFFF.800E2000</td>
<td>FFFFFFF.800E5C00</td>
</tr>
<tr>
<td>PROCESS_MANAGEMENT_MON</td>
<td>FFFFFFF.800E6000</td>
<td>FFFFFFF.8010B000</td>
</tr>
<tr>
<td>SYS$VM</td>
<td>FFFFFFF.8010C000</td>
<td>FFFFFFF.80167200</td>
</tr>
<tr>
<td>SHELL$K</td>
<td>FFFFFFF.80168000</td>
<td>FFFFFFF.80169200</td>
</tr>
<tr>
<td>LOCKING</td>
<td>FFFFFFF.8016A000</td>
<td>FFFFFFF.8017BE00</td>
</tr>
<tr>
<td>MESSAGE_Routines</td>
<td>FFFFFFF.8017C000</td>
<td>FFFFFFF.80182A00</td>
</tr>
<tr>
<td>LOGICAL_NAMES</td>
<td>FFFFFFF.8018A000</td>
<td>FFFFFFF.80186C00</td>
</tr>
<tr>
<td>F11BKQP</td>
<td>FFFFFFF.80194000</td>
<td>FFFFFFF.80197A00</td>
</tr>
<tr>
<td>IMAGE_MANAGEMENT</td>
<td>FFFFFFF.80198000</td>
<td>FFFFFFF.801A0E00</td>
</tr>
<tr>
<td>SECURITY</td>
<td>FFFFFFF.801A2000</td>
<td>FFFFFFF.801A3A00</td>
</tr>
<tr>
<td>SYSGETSYI</td>
<td>FFFFFFF.801A4000</td>
<td>FFFFFFF.801C5000</td>
</tr>
<tr>
<td>SYS$TRANSACTION_Services</td>
<td>FFFFFFF.801C6000</td>
<td>FFFFFFF.801C7000</td>
</tr>
<tr>
<td>SYS$VCC_MON</td>
<td>FFFFFFF.80160000</td>
<td>FFFFFFF.80162C00</td>
</tr>
<tr>
<td>SYS$IPC_Services</td>
<td>FFFFFFF.8016D000</td>
<td>FFFFFFF.80171A00</td>
</tr>
<tr>
<td>SYS$TCP_Dyn</td>
<td>FFFFFFF.80180000</td>
<td>FFFFFFF.80182000</td>
</tr>
<tr>
<td>SYS$MMEM_Services</td>
<td>FFFFFFF.80201000</td>
<td>FFFFFFF.80218000</td>
</tr>
<tr>
<td>SYS$TITDriver</td>
<td>FFFFFFF.8021C000</td>
<td>FFFFFFF.8022E000</td>
</tr>
<tr>
<td>SYS$PKCDriver</td>
<td>FFFFFFF.80230000</td>
<td>FFFFFFF.80240000</td>
</tr>
<tr>
<td>SYS$OKDriver</td>
<td>FFFFFFF.80242000</td>
<td>FFFFFFF.80251600</td>
</tr>
<tr>
<td>RMS</td>
<td>FFFFFFF.80252000</td>
<td>FFFFFFF.8025E000</td>
</tr>
<tr>
<td>SYS$GADriver</td>
<td>FFFFFFF.80260000</td>
<td>FFFFFFF.8026E000</td>
</tr>
<tr>
<td>SYS$ECDriver</td>
<td>FFFFFFF.8027E000</td>
<td>FFFFFFF.80281000</td>
</tr>
<tr>
<td>SYS$LAN</td>
<td>FFFFFFF.8028D000</td>
<td>FFFFFFF.8028E000</td>
</tr>
<tr>
<td>SYS$LAN_CSMACD</td>
<td>FFFFFFF.802DA000</td>
<td>FFFFFFF.802E6000</td>
</tr>
<tr>
<td>SYS$MKDriver</td>
<td>FFFFFFF.802E8000</td>
<td>FFFFFFF.802F1C00</td>
</tr>
<tr>
<td>SYS$DRDriver</td>
<td>FFFFFFF.802F2000</td>
<td>FFFFFFF.802F9600</td>
</tr>
<tr>
<td>SYS$SODriver</td>
<td>FFFFFFF.802FA000</td>
<td>FFFFFFF.802FF000</td>
</tr>
<tr>
<td>SYS$INDriver</td>
<td>FFFFFFF.80300000</td>
<td>FFFFFFF.80308A00</td>
</tr>
<tr>
<td>NEHDriver</td>
<td>FFFFFFF.8030C000</td>
<td>FFFFFFF.80310200</td>
</tr>
<tr>
<td>NEHDriver</td>
<td>FFFFFFF.80312000</td>
<td>FFFFFFF.80329E00</td>
</tr>
<tr>
<td>SYS$IMDriver</td>
<td>FFFFFFF.8032A000</td>
<td>FFFFFFF.8032E000</td>
</tr>
<tr>
<td>SYS$IKDriver</td>
<td>FFFFFFF.80330000</td>
<td>FFFFFFF.8033AC00</td>
</tr>
<tr>
<td>NDRDriver</td>
<td>FFFFFFF.8033C000</td>
<td>FFFFFFF.8033F800</td>
</tr>
<tr>
<td>SYS$MSDriver</td>
<td>FFFFFFF.80340000</td>
<td>FFFFFFF.80341600</td>
</tr>
<tr>
<td>SYS$CTDriver</td>
<td>FFFFFFF.80342000</td>
<td>FFFFFFF.8034D200</td>
</tr>
<tr>
<td>SYS$RITDriver</td>
<td>FFFFFFF.8034E000</td>
<td>FFFFFFF.80351800</td>
</tr>
<tr>
<td>SYS$FTDriver</td>
<td>FFFFFFF.80352000</td>
<td>FFFFFFF.80354200</td>
</tr>
</tbody>
</table>

### Execlet Data Region

<table>
<thead>
<tr>
<th>Base</th>
<th>End</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFFF.80C000000</td>
<td>FFFFFFF.80CC0000</td>
<td>96/1536</td>
</tr>
<tr>
<td>FFFFFFF.80000000</td>
<td>FFFFFFF.800000FC</td>
<td>Free/11</td>
</tr>
<tr>
<td>0.7 MB</td>
<td>1525</td>
<td></td>
</tr>
<tr>
<td>FFFFFFF.80D17D00</td>
<td>0.000000000000000</td>
<td>Initial Size 128/2048</td>
</tr>
<tr>
<td>FFFFFFF.80000000</td>
<td>FFFFFFF.80000000</td>
<td>000000200</td>
</tr>
<tr>
<td>FFFFFFF.80000000</td>
<td>FFFFFFF.80000000</td>
<td>0000005F5</td>
</tr>
</tbody>
</table>
SDA CLUE Extension Commands

CLUE MEMORY

<table>
<thead>
<tr>
<th>Image</th>
<th>Base</th>
<th>End</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS$PUBLIC_VECTORS</td>
<td>FFFFFFFF.80C00000</td>
<td>FFFFFFFF.80C05000</td>
<td>00000500</td>
</tr>
<tr>
<td>SYS$BASE_IMAGE</td>
<td>FFFFFFFF.80C05000</td>
<td>FFFFFFFF.80C25E00</td>
<td>00020E00</td>
</tr>
<tr>
<td>SYS$CNBDTRIVER</td>
<td>FFFFFFFF.80C25E00</td>
<td>FFFFFFFF.80C26200</td>
<td>00000400</td>
</tr>
<tr>
<td>SYS$SMSCA_BTDTRIVER</td>
<td>FFFFFFFF.80C26200</td>
<td>FFFFFFFF.80C29400</td>
<td>00003200</td>
</tr>
<tr>
<td>SYS$ESBTDRIVER</td>
<td>FFFFFFFF.80C29400</td>
<td>FFFFFFFF.80C29800</td>
<td>00000400</td>
</tr>
<tr>
<td>SYS$OPDRIVER</td>
<td>FFFFFFFF.80C29800</td>
<td>FFFFFFFF.80C2A200</td>
<td>0000A000</td>
</tr>
<tr>
<td>SYSTEM_DEBUG</td>
<td>FFFFFFFF.80C2A200</td>
<td>FFFFFFFF.80C4E400</td>
<td>00024200</td>
</tr>
<tr>
<td>SYSTEM_PRIMITIVES</td>
<td>FFFFFFFF.80C4E400</td>
<td>FFFFFFFF.80C58200</td>
<td>0009E00</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION</td>
<td>FFFFFFFF.80C58200</td>
<td>FFFFFFFF.80C5A000</td>
<td>0001E00</td>
</tr>
<tr>
<td>ERRORLOG</td>
<td>FFFFFFFF.80C5A000</td>
<td>FFFFFFFF.80C5A600</td>
<td>0000600</td>
</tr>
<tr>
<td>SYS$CPU_ROUTINES_0402</td>
<td>FFFFFFFF.80C5A600</td>
<td>FFFFFFFF.80C5CA00</td>
<td>00002400</td>
</tr>
<tr>
<td>EXCEPTION_MON</td>
<td>FFFFFFFF.80C5CA00</td>
<td>FFFFFFFF.80C64C00</td>
<td>00008200</td>
</tr>
<tr>
<td>IO_ROUTINES_MON</td>
<td>FFFFFFFF.80C64C00</td>
<td>FFFFFFFF.80C66A00</td>
<td>00005E00</td>
</tr>
<tr>
<td>SYSDevice</td>
<td>FFFFFFFF.80C66A00</td>
<td>FFFFFFFF.80C6B600</td>
<td>00000C00</td>
</tr>
<tr>
<td>PROCESS_MANAGEMENT_MON</td>
<td>FFFFFFFF.80C6B600</td>
<td>FFFFFFFF.80C72600</td>
<td>00007000</td>
</tr>
<tr>
<td>SYS$VM</td>
<td>FFFFFFFF.80C72600</td>
<td>FFFFFFFF.80C79000</td>
<td>00006A00</td>
</tr>
<tr>
<td>SHELL8K</td>
<td>FFFFFFFF.80C79000</td>
<td>FFFFFFFF.80C7A000</td>
<td>00010D00</td>
</tr>
<tr>
<td>LOCKING</td>
<td>FFFFFFFF.80C7A000</td>
<td>FFFFFFFF.80C7A600</td>
<td>0001A00</td>
</tr>
<tr>
<td>MESSAGE_ROUTINES</td>
<td>FFFFFFFF.80C7A600</td>
<td>FFFFFFFF.80C7D000</td>
<td>0001600</td>
</tr>
<tr>
<td>LOGICAL_NAMES</td>
<td>FFFFFFFF.80C7D000</td>
<td>FFFFFFFF.80C7D200</td>
<td>0001200</td>
</tr>
<tr>
<td>F11BQXP</td>
<td>FFFFFFFF.80C7D200</td>
<td>FFFFFFFF.80C7F000</td>
<td>0001800</td>
</tr>
<tr>
<td>SYS/LICENSE</td>
<td>FFFFFFFF.80C7F000</td>
<td>FFFFFFFF.80C7F200</td>
<td>0000400</td>
</tr>
<tr>
<td>IMAGE_MANAGEMENT</td>
<td>FFFFFFFF.80C7F200</td>
<td>FFFFFFFF.80C80600</td>
<td>0000800</td>
</tr>
<tr>
<td>SECURITY</td>
<td>FFFFFFFF.80C80600</td>
<td>FFFFFFFF.80C83000</td>
<td>0002A00</td>
</tr>
<tr>
<td>SYS$GETSYI</td>
<td>FFFFFFFF.80C83000</td>
<td>FFFFFFFF.80C83200</td>
<td>0000200</td>
</tr>
<tr>
<td>SYS$TRANSACTION_SERVICES</td>
<td>FFFFFFFF.80C83200</td>
<td>FFFFFFFF.80C89E00</td>
<td>0006C00</td>
</tr>
<tr>
<td>SYS$UTC_SERVICES</td>
<td>FFFFFFFF.80C89E00</td>
<td>FFFFFFFF.80C8A200</td>
<td>0000400</td>
</tr>
<tr>
<td>SYS$VCC_MON</td>
<td>FFFFFFFF.80C8A200</td>
<td>FFFFFFFF.80C8BC00</td>
<td>0001A00</td>
</tr>
<tr>
<td>SYS$IPC_SERVICES</td>
<td>FFFFFFFF.80C8BC00</td>
<td>FFFFFFFF.80C91000</td>
<td>0005400</td>
</tr>
<tr>
<td>SYS$LD_DYN</td>
<td>FFFFFFFF.80C91000</td>
<td>FFFFFFFF.80C92200</td>
<td>0001200</td>
</tr>
<tr>
<td>SYS$MEM_SERVICES</td>
<td>FFFFFFFF.80C92200</td>
<td>FFFFFFFF.80C92600</td>
<td>0000400</td>
</tr>
<tr>
<td>SYS$TTRDRIVER</td>
<td>FFFFFFFF.80C92600</td>
<td>FFFFFFFF.80C94C00</td>
<td>0002600</td>
</tr>
<tr>
<td>SYS$PKCDRIVER</td>
<td>FFFFFFFF.80C94C00</td>
<td>FFFFFFFF.80C96A00</td>
<td>0001E00</td>
</tr>
<tr>
<td>SYS$DKDRIVER</td>
<td>FFFFFFFF.80C96A00</td>
<td>FFFFFFFF.80C99800</td>
<td>0002E00</td>
</tr>
<tr>
<td>RMS</td>
<td>FFFFFFFF.80C99800</td>
<td>FFFFFFFF.80CAC000</td>
<td>0001400</td>
</tr>
<tr>
<td>RECOVERY_UNIT_SERVICES</td>
<td>FFFFFFFF.80CAC000</td>
<td>FFFFFFFF.80CB000</td>
<td>0000400</td>
</tr>
<tr>
<td>SYS$GADRIVER</td>
<td>FFFFFFFF.80CB000</td>
<td>FFFFFFFF.80CAFC00</td>
<td>0000400</td>
</tr>
<tr>
<td>SYS$ECDRIVER</td>
<td>FFFFFFFF.80CAFC00</td>
<td>FFFFFFFF.80CB000</td>
<td>0000400</td>
</tr>
<tr>
<td>SYS$LAN</td>
<td>FFFFFFFF.80CB000</td>
<td>FFFFFFFF.80CB0800</td>
<td>00001000</td>
</tr>
<tr>
<td>SYS$LAN_CSMACD</td>
<td>FFFFFFFF.80CB0800</td>
<td>FFFFFFFF.80CB1800</td>
<td>00001000</td>
</tr>
<tr>
<td>SYS$MKDRIVER</td>
<td>FFFFFFFF.80CB1800</td>
<td>FFFFFFFF.80CB3000</td>
<td>0001800</td>
</tr>
<tr>
<td>SYS$YRDTRIVER</td>
<td>FFFFFFFF.80CB3000</td>
<td>FFFFFFFF.80CB3C00</td>
<td>0000C00</td>
</tr>
<tr>
<td>SYS$SODRIVER</td>
<td>FFFFFFFF.80CB3C00</td>
<td>FFFFFFFF.80CB4E00</td>
<td>0001200</td>
</tr>
<tr>
<td>SYS$INDRIVER</td>
<td>FFFFFFFF.80CB4E00</td>
<td>FFFFFFFF.80CB5E00</td>
<td>0001000</td>
</tr>
<tr>
<td>NETDRIVER</td>
<td>FFFFFFFF.80CB5E00</td>
<td>FFFFFFFF.80CB8800</td>
<td>0002A00</td>
</tr>
<tr>
<td>SYS$IMDRIVER</td>
<td>FFFFFFFF.80CB8800</td>
<td>FFFFFFFF.80CB9400</td>
<td>0000C00</td>
</tr>
<tr>
<td>SYS$IKDRIVER</td>
<td>FFFFFFFF.80CB9400</td>
<td>FFFFFFFF.80CBA000</td>
<td>0001600</td>
</tr>
<tr>
<td>NDORIVER</td>
<td>FFFFFFFF.80CBA000</td>
<td>FFFFFFFF.80CBB400</td>
<td>0000A00</td>
</tr>
<tr>
<td>SYS$WSDRIVER</td>
<td>FFFFFFFF.80CBB400</td>
<td>FFFFFFFF.80C8B000</td>
<td>0000800</td>
</tr>
<tr>
<td>SYS$CTDRIVER</td>
<td>FFFFFFFF.80C8B000</td>
<td>FFFFFFFF.80C8B000</td>
<td>0001C00</td>
</tr>
<tr>
<td>SYS$RTTDRIVER</td>
<td>FFFFFFFF.80C8B000</td>
<td>FFFFFFFF.80C8B000</td>
<td>0000A00</td>
</tr>
<tr>
<td>SYS$FTPDRIVER</td>
<td>FFFFFFFF.80C8B000</td>
<td>FFFFFFFF.80C8B000</td>
<td>0000800</td>
</tr>
<tr>
<td>11 free Slices</td>
<td>FFFFFFFF.80C8B000</td>
<td>FFFFFFFF.80C8B000</td>
<td>0001600</td>
</tr>
</tbody>
</table>

5–20 SDA CLUE Extension Commands
The CLUE MEMORY/GH/FULL command displays data structures that describe granularity hint regions and huge pages.
4. SDA> CLUE MEMORY/LAYOUT

System Virtual Address Space Layout:

<table>
<thead>
<tr>
<th>Item</th>
<th>Base</th>
<th>End</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Virtual Base Address</td>
<td>FFFFFEFE.00000000</td>
<td>FFFFFEFE.00280000</td>
<td>00280000</td>
</tr>
<tr>
<td>PFN Database</td>
<td>FFFFFEFE.00280000</td>
<td>FFFFFEFE.0089CD38</td>
<td>0061AD38</td>
</tr>
<tr>
<td>Permanent Mapping of System LIPT</td>
<td>FFFFFF6.6FC1A0000</td>
<td>FFFFFF6.6FC220000</td>
<td>00080000</td>
</tr>
<tr>
<td>Global Page Table (GPT)</td>
<td>FFFFFF6.6FC220000</td>
<td>FFFFFF7.70000000</td>
<td>003DE000</td>
</tr>
<tr>
<td>Resource Hash Table</td>
<td>FFFFFF6.6FC220000</td>
<td>FFFFFF7.70000000</td>
<td>003DE000</td>
</tr>
<tr>
<td>Lock ID Table</td>
<td>FFFFFF6.6FC220000</td>
<td>FFFFFF7.70000000</td>
<td>003DE000</td>
</tr>
<tr>
<td>ExecLct Code Region</td>
<td>FFFFFF8.80000000</td>
<td>FFFFFF8.80800000</td>
<td>00800000</td>
</tr>
<tr>
<td>Resident Image Code Region</td>
<td>FFFFFF8.80800000</td>
<td>FFFFFF8.81000000</td>
<td>00800000</td>
</tr>
<tr>
<td>System Header</td>
<td>FFFFFF8.81400000</td>
<td>FFFFFF8.8140E0000</td>
<td>000E0000</td>
</tr>
<tr>
<td>Error Log Allocation Buffers</td>
<td>FFFFFF8.8140E0000</td>
<td>FFFFFF8.81414000</td>
<td>00060000</td>
</tr>
<tr>
<td>Nonpaged Pool (initial size)</td>
<td>FFFFFF8.81414000</td>
<td>FFFFFF8.817C8000</td>
<td>003B4000</td>
</tr>
<tr>
<td>Nonpaged Pool Expansion Area</td>
<td>FFFFFF8.817C8000</td>
<td>FFFFFF8.82664000</td>
<td>00E9C000</td>
</tr>
<tr>
<td>ExecLct Data Region</td>
<td>FFFFFF8.81000000</td>
<td>FFFFFF8.81400000</td>
<td>00400000</td>
</tr>
<tr>
<td>Fork Buffers Secondary to Primary</td>
<td>FFFFFF8.8268C0000</td>
<td>FFFFFF8.8268E0000</td>
<td>00020000</td>
</tr>
<tr>
<td>Erase Pattern Buffer Page</td>
<td>FFFFFF8.8268E0000</td>
<td>FFFFFF8.82690000</td>
<td>00020000</td>
</tr>
<tr>
<td>363 Balance Slots, 33 pages each</td>
<td>FFFFFF8.826A0000</td>
<td>FFFFFF8.84360000</td>
<td>05D96000</td>
</tr>
<tr>
<td>Paged Pool</td>
<td>FFFFFF8.84360000</td>
<td>FFFFFF8.887E4000</td>
<td>003AE000</td>
</tr>
<tr>
<td>System Control Block (SCB)</td>
<td>FFFFFF8.887E4000</td>
<td>FFFFFF8.887EC0000</td>
<td>00080000</td>
</tr>
<tr>
<td>Restart Parameter Block (HWRPB)</td>
<td>FFFFFF8.88832000</td>
<td>FFFFFF8.8832B48</td>
<td>0000B48</td>
</tr>
<tr>
<td>Erase Pattern Page Table Page</td>
<td>FFFFFF8.82690000</td>
<td>FFFFFF8.82692000</td>
<td>00020000</td>
</tr>
<tr>
<td>Posix Cloning Parent Page Mapping</td>
<td>FFFFFF8.8B1E0000</td>
<td>FFFFFF8.8B200000</td>
<td>00020000</td>
</tr>
<tr>
<td>Posix Cloning Child Page Mapping</td>
<td>FFFFFF8.8B200000</td>
<td>FFFFFF8.8B220000</td>
<td>00020000</td>
</tr>
<tr>
<td>Swapper Process Kernel Stack</td>
<td>FFFFFF8.8B560000</td>
<td>FFFFFF8.8B5A0000</td>
<td>00040000</td>
</tr>
<tr>
<td>Swapper Map</td>
<td>FFFFFF8.8B600000</td>
<td>FFFFFF8.8B820000</td>
<td>00220000</td>
</tr>
<tr>
<td>Idle Loop’s Mapping of Zero Pages</td>
<td>FFFFFF8.8C5E0000</td>
<td>FFFFFF8.8C600000</td>
<td>00020000</td>
</tr>
<tr>
<td>PrimCPU Machine Check Logout Area</td>
<td>FFFFFF8.8C504000</td>
<td>FFFFFF8.8C608000</td>
<td>00004000</td>
</tr>
<tr>
<td>PrimCPU Sys Context Kernel Stack</td>
<td>FFFFFF8.8C580000</td>
<td>FFFFFF8.8C5C0000</td>
<td>00040000</td>
</tr>
<tr>
<td>Tape Mount Verification Buffer</td>
<td>FFFFFF8.8C620000</td>
<td>FFFFFF8.8C660000</td>
<td>00040000</td>
</tr>
<tr>
<td>Mount Verification Buffer</td>
<td>FFFFFF8.8C660000</td>
<td>FFFFFF8.8C680000</td>
<td>00020000</td>
</tr>
<tr>
<td>Demand Zero Optimization Page</td>
<td>FFFFFF8.8E680000</td>
<td>FFFFFF8.8E6A0000</td>
<td>00020000</td>
</tr>
<tr>
<td>Executive Mode Data Page</td>
<td>FFFFFF8.8E6A0000</td>
<td>FFFFFF8.8E6C0000</td>
<td>00020000</td>
</tr>
<tr>
<td>System Space Expansion Region</td>
<td>FFFFFF8.8C000000</td>
<td>FFFFFF8.8FD000000</td>
<td>73DF0000</td>
</tr>
<tr>
<td>System Page Table Window</td>
<td>FFFFFF8.8FD000000</td>
<td>FFFFFF8.8FD000000</td>
<td>00200000</td>
</tr>
<tr>
<td>N/A Space</td>
<td>FFFFFF8.8FD000000</td>
<td>FFFFFF8.8FD000000</td>
<td>00100000</td>
</tr>
</tbody>
</table>

The CLUE MEMORY/LAYOUT command decodes and displays the system virtual address space layout.
The CLUE MEMORY/LOOKASIDE command summarizes the state of nonpageable lookaside lists. For each list, an indication of whether the queue is well formed is given. If a queue is not well formed or is invalid, messages indicating what is wrong with the queue are displayed. This command is analogous to the SDA command VALIDATE QUEUE.

These messages can also appear frequently when you use the VALIDATE QUEUE command within an SDA session that is analyzing a running system. In a running system, the composition of a queue can change while the command is tracing its links, thus producing an error message.
The CLUE MEMORY/STATISTIC command displays systemwide performance data such as page fault, I/O, pool, lock manager, MSCP, and file system cache statistics.
CLUE PROCESS

Displays process-related information from the current process context.

Format

CLUE PROCESS [/qualifier[,...]]

Parameters

None.

Qualifiers

/BUFFER [/ALL]
Displays the buffer objects for the current process. If the /ALL qualifier is specified, then the buffer objects for all processes (that is, all existing buffer objects) are displayed.

/LAYOUT
Displays the process P1 virtual address space layout.

/LOGICAL
Displays the process logical names and equivalence names, if they can be accessed.

/RECALL
Displays the DCL recall buffer, if it can be accessed.

Description

The CLUE PROCESS command displays process-related information from the current process context. Much of this information is in pageable address space and thus may not be present in a dump file.

Examples

1. SDA> CLUE PROCESS/LOGICAL
   Process Logical Names:
   ----------------------
   "$output" = "CLAWS$LTA5004:"
   "$output" = "CLAWS$LTA5004:"
   "$disk" = "WORK1:" 
   "$backup_file" = "$65$DUA6"
   "$putmsg" = "...À...À...
   "$command" = "CLAWS$LTA5004:"
   "$tape_logical_name" = "$1$MUA3:
   "$TT" = "$TA5004:
   "$input" = "$65$DUA6:"
   "$input" = "CLAWS$LTA5004:"
   "$error" = "21C00303.LOG"
   "$error" = "CLAWS$LTA5004:"
   "$error_file" = "$65$DUA6"

The CLUE PROCESS/LOGICAL command displays logical names for each running process.
The CLUE PROCESS/RECALL command displays a listing of the DCL commands that have been executed most recently.
CLUE REGISTER

Displays the active register set for the crash CPU. The CLUE REGISTER command is valid only when analyzing crash dumps.

Format

CLUE REGISTER

Parameters

None.

Qualifiers

None.

Description

The CLUE REGISTER command displays the active register set of the crash CPU. It also identifies any known data structures, symbolizes any system virtual addresses, interprets the processor status (PS), and attempts to interpret R0 as a condition code.

Example

SDA> CLUE REGISTER
Current Registers: Process index: 0042 Process name: BATCH_3 PCB: 817660C0 (CPU 1)
---------------------------------------------------------------------------------------------------------------------
    R0 = 00000000.00000000    R1 = FFFFFFFF.814A2C80    MP_CPU (CPU Id 1)
    R2 = 00000000.00000000    R3 = 00000000.23D6BBEE
    R4 = 00000000.00000064    R5 = FFFFFFFF.831F8000    PHDR
    R6 = 00000000.12F75475    R7 = 00000000.010C7A70
    R8 = 00000000.00000001    R9 = 00000000.00000000
    R10 = 00000000.00000000   R11 = FFFFFFFF.814A2C80    MP_CPU (CPU Id 1)
    R12 = FFFFFFFF.810AA5E0    SYSTEM_SYNCHRONIZATION+293E0
    R13 = FFFFFFFF.810AC408    SMP$TIMEOUT
    R14 = FFFFFFFF.810AED00    SMP$GL_SCHED
    R15 = 00000000.7FPA1DD8    R16 = 00000000.0000078C
    R17 = 00000000.00000000   R18 = FFFFFFFF.810356C0    SYS$CPU_ROUTINES_2208+1D6C0
    R19 = FFFFFFFF.81006000    EXE$GR_SYSTEM_DATA.Cells
    R20 = FFFFFFFF.80120F00    SCH$QEND_C+00080
    R21 = 00000000.00000000   R22 = FFFFFFFF.00000000
    R23 = 00000000.00000000   R24 = 00000000.00000000
    A1 = FFFFFFFF.81006000    EXE$GR_SYSTEM_DATA.Cells
    RA = 00000000.00000000    PV = 00000000.00000000
    R28 = FFFFFFFF.810194A0    EXE$GL_TIME_CONTROL

SDA CLUE Extension Commands 5–27
SDA CLUE Extension Commands

CLUE REGISTER

FP = 00000000.7FFA1F90
PC = FFFFFFFF.800863A8  SMP$TIMEOUT_C+00068
PS = 18000000.00000804  Kernel Mode, IPL 8, Interrupt
**CLUE SG**

Displays the scatter-gather map.

**Format**

```
CLUE SG [/CRAB=address]
```

**Parameters**

None.

**Qualifier**

```
/CRAB=address
```

Displays the ringbuffer for the specified Counted Resource Allocation Block (CRAB). The default action is to display the ringbuffer for all CRABs.

**Description**

CLUE SG decodes and displays the scatter/gather ringbuffer entries.

**Examples**

1. `SDA> CLUE SG/CRAB=81224740`
   Scatter/Gather Ringbuffer for CRAB 81224740:
   
<table>
<thead>
<tr>
<th>XAct</th>
<th>CRCTX</th>
<th>Item_Num</th>
<th>Item_Cnt</th>
<th>DMA_Addr</th>
<th>Status</th>
<th>Callers_PC</th>
<th>Count</th>
<th>Buf_Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLO</td>
<td>81272780</td>
<td>00000020 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000018</td>
<td>81240AE0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272700</td>
<td>0000001C 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000017</td>
<td>81240AC0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272680</td>
<td>00000018 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000016</td>
<td>81240AA0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272600</td>
<td>00000014 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000015</td>
<td>81240A80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272580</td>
<td>00000010 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000014</td>
<td>81240A60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272500</td>
<td>0000000C 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000013</td>
<td>81240A40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272480</td>
<td>00000008 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000012</td>
<td>81240A20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLO</td>
<td>81272400</td>
<td>00000004 00000004 00000000 00000001</td>
<td>847DA94</td>
<td>SYSENDRIVER+01A94</td>
<td>00000011</td>
<td>81240A00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   In this example, the scatter-gather ringbuffer for the CRAB at address 81224740 is displayed.

2. `SDA> CLUE SG/CRAB=8120D600`
   Scatter/Gather Ringbuffer for CRAB 8120D600:
   
<table>
<thead>
<tr>
<th>XAct</th>
<th>CRCTX</th>
<th>Item_Num</th>
<th>Item_Cnt</th>
<th>DMA_Addr</th>
<th>Status</th>
<th>Callers_PC</th>
<th>Count</th>
<th>Buf_Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLO</td>
<td>8128A380</td>
<td>00001C00 00004000 00000000 00000001</td>
<td>8480E990</td>
<td>SYSENDRIVER+02990</td>
<td>00000000</td>
<td>8121C760</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   In this example, the scatter-gather ringbuffer for the CRAB address 8120D600 is displayed.
CLUE STACK

Identifies and displays the current stack. Use the SDA command SHOW STACK to display and decode the whole stack for the more common bugcheck types.

Format

CLUE STACK

Parameters

None.

Qualifiers

None.

Description

The CLUE STACK command identifies and displays the current stack together with the upper and lower stack limits. In case of a FATALEXCPT, INVECEPTN, SSRVEXCEPT, UNXSIGNAL, or PGFIPLHI bugcheck, CLUE STACK tries to decode the whole stack.

Example

SDA> CLUE STACK
Stack Decoder:
----------
Normal Process Kernel Stack:
Stack Pointer 00000000.7FFA1C98
Stack Limits (low) 00000000.7FFA0000
(high) 00000000.7FFA2000
SSRVEXCEPT Stack:
------------------
Stack Pointer SP => 00000000.7FFA1C98
Information saved by Bugcheck:
a(Signal Array) 00000000.7FFA1C98 00000000.00000000
EXE$EXCPTN[E] Temporary Storage:
EXE$EXCPTN[E] Stack Frame:
PV 00000000.7FFA1CA0 FFFFFFFF.829CF010 EXE$EXCPTN
Entry Point FFFFFFFF.82A21000 EXE$EXCPTN_C
return PC 00000000.7FFA1CA8 FFFFFFFF.82A2059C SYS$CALL_HANDL_C+0002C
saved R2 00000000.7FFA1CB0 00000000.00000000
saved FP 00000000.7FFA1CB8 00000000.7FFA1CD0
SYS$CALL_HANDL Temporary Storage:
00000000.7FFA1CC0 FFFFFFFF.829CEDA8 SYS$CALL_HANDL
00000000.7FFA1CC8 00000000.00000000
SYS$CALL_HANDL Stack Frame:
PV 00000000.7FFA1CD0 FFFFFFFF.829CEDA8 SYS$CALL_HANDL
Entry Point FFFFFFFF.82A20570 SYS$CALL_HANDL_C
return PC 00000000.7FFA1CD8 FFFFFFFF.82A1E930 CHF_REI+000DC
saved FP 00000000.7FFA1CE8 00000000.7FFA1F40

5–30 SDA CLUE Extension Commands
Fixed Exception Context Area:
Linkage Pointer 0000000.7FFA1CF00 0000000.7FFA1CF00 EXCEPTION_MON_NPRW+06D80
a(Signal Array) 00000000.7FFA1CF8 00000000.7FFA1CF8
a(Mechanism Array) 00000000.7FFA1D00 00000000.7FFA1D40
a(Exception Frame) 00000000.7FFA1D08 00000000.7FFA1F00
Exception FP 00000000.7FFA1D10 00000000.7FFA1F40
Unwind SP 00000000.7FFA1D18 00000000.00000000
Reinvokable FP 00000000.7FFA1D20 00000000.00000000
Unwind Target 00000000.7FFA1D28 00000000.00020000 SYSSK_VERSION_04
#Sig Args/Byte Cnt 00000000.7FFA1D30 00000000.00020050 BUGS_NETCVPKT
a(Msg)/Final Status 00000000.7FFA1D38 829CE050.000008F8 BUGS_SEQ_NUM_OVF

Mechanism Array:
Flags/Arguments 00000000.7FFA1D40 00000000.0000002C
a(Establisher FP) 00000000.7FFA1D48 00000000.7AFFBAD0
reserved/Depth 00000000.7FFA1D50 FFFFFFFF.FFFFFFFD
a(Handler Data) 00000000.7FFA1D58 00000000.00000000
a(Exception Frame) 00000000.7FFA1D60 00000000.7FFA1F00
a(Signal Array) 00000000.7FFA1D68 00000000.7FFA1EB8
saved R0 00000000.7FFA1D70 00000000.00020000 SYSSK_VERSION_04
saved R1 00000000.7FFA1D78 00000000.00000000
saved R16 00000000.7FFA1D80 00000000.00020004 UCSM_NI_PRM_MLT+00004
saved R17 00000000.7FFA1D88 00000000.00010000 SYSSK_VERSION_16+00010
saved R18 00000000.7FFA1D90 FFFFFFFF.FFFFFFFF
saved R19 00000000.7FFA1D98 00000000.00000000
saved R20 00000000.7FFA1DA0 00000000.7FFA1F50
saved R21 00000000.7FFA1DA8 00000000.00000000
saved R22 00000000.7FFA1DB0 00000000.00010050 SYSSK_VERSION_16+0010
saved R23 00000000.7FFA1DB8 00000000.00000000
saved R24 00000000.7FFA1DC0 00000000.00010051 SYSSK_VERSION_16+0011
saved R25 00000000.7FFA1DC8 00000000.00000000
saved R26 00000000.7FFA1DD0 FFFFFFFF.8010ACA4 AMAC$EMUL_CALL_NATIVE_C+000A
saved R27 00000000.7FFA1DD8 00000000.00010050 SYSSK_VERSION_16+0010
saved R28 00000000.7FFA1DE0 00000000.00000000
FP Regs not valid [...............

SP Align = 10(hex) [...............

Signal Array:
Arguments 00000000.7FFA1EB8 00000000.7FFA1EB8
Condition 00000000.7FFA1EBC 00000000.7FFA1EBC
Argument #2 00000000.7FFA1EC0 00010000 LDRIMG$M_NPAGED_LOAD
Argument #3 00000000.7FFA1EC4 00000000
Argument #4 00000000.7FFA1EC8 00030078 SYSSK_VERSION_01+00078
Argument #5 00000000.7FFA1ECC 00000003

64-bit Signal Array:
Arguments 00000000.7FFA1E0 00002604.00000005
Condition 00000000.7FFA1E8 00000000.0000000C
Argument #2 00000000.7FFA1EE0 00000000.00010000 LDRIMG$M_NPAGED_LOAD
Argument #3 00000000.7FFA1EE8 00000000.00000000
Argument #4 00000000.7FFA1EF0 00030078 SYSSK_VERSION_01+00078
Argument #5 00000000.7FFA1EFF 00000000.00000003

Interrupt/Exception Frame:
saved R2 00000000.7FFA1F00 00000000.00000003
saved R3 00000000.7FFA1F08 FFFFFFFF.80C63460 EXCEPTION_MON_NPRW+06A60
saved R4 00000000.7FFA1F10 FFFFFFFF.80D12740 PCB
saved R5 00000000.7FFA1F18 00000000.000000C8
saved R6 00000000.7FFA1F20 00000000.00030038 SYSSK_VERSION_01+00038
saved R7 00000000.7FFA1F28 00000000.7FFA1FC0
saved PC 00000000.7FFA1F30 00000000.00030078 SYSSK_VERSION_01+00078
saved PS 00000000.7FFA1F38 00000000.00000003 IPL INT CURR PREV

SP Align = 00(hex) [...............

00 0 Kern User
CLUE STACK identifies and displays the current stack and its upper and lower limit. It then decodes the current stack if it is one of the more common bugcheck types. In this case, CLUE STACK tries to decode the entire INVEXCEPTN stack.
CLUE SYSTEM

Displays the contents of the shared logical name tables in the system.

Format

CLUE SYSTEM /LOGICAL

Parameters

None.

Qualifier

/LOGICAL
Displays all the shared logical names.

Description

The CLUE SYSTEM/LOGICAL command displays the contents of the shared logical name tables in the system.

Example

SDA> CLUE SYSTEM/LOGICAL
Shareable Logical Names:
------------------------
"XMICONBMSEARCHPATH" = "CDE$HOME_DEFAULTS:[ICONS]%B%M.BM"
"MTHRTL TV" = "MTHRTL_D53 TV"
"SMGSHR TV" = "SMGSHR"
"DEC$DEFAULT_KEYBOARD_MAP" = "NORTH_AMERICAN_LK401AA"
"CONVSHR TV" = "CONVSHR"
"XDP$INCLUDE" = "SYS$SYSROOT:[XDP$INCLUDE]"
"DEC$SYSTEM_DEFAULTS" = "SYS$SYSROOT:[DEC$DEFAULTS.USER]"
"SYS$PS_FONT_METRICS" = "SYS$SYSROOT:[SYSPS.FONT_METRICS.USER]"
"SYSTIMEZONE_NAME" = "???
"STARTUP$STARTUP_VMS" = "SYS$STARTUP:VMS$VMS.DAT"
"PASMCG" = "PAS$MSG"
"UCX$HOST" = "SYS$COMMON:[SYSEXEB]UCX$HOST.DAT;1"
"SYSSYLOGIN" = "SYS$MANAGER:SYLOGIN"
"DNS$SYSTEM" = "DNS$SYSTEM_TABLE"
"IPC$ACP_ERRMBX" = "d.0."
"CDE$DETACHED_LOGICALS" = "DEC$DISPLAY,LANG"
"DEC$SERVER SCREENS" = "GXA0"
"DNS$_COTOAD_MBX" = "b.0."
"DNS$LOGICAL" = "DNS$SYSTEM"
"OSIT$MAILBOX" = "b.0."
"XNL$SHR_TV" = "XNL$SHR_TV_SUPPORT.EXE"
"MOM$SYSTEM" = "SYS$SYSROOT:[MOM$SYSTEM]"
"MOP$LOAD" = "SYS$SYSROOT:<MOM$SYSTEM>"
CLUE VCC

Displays virtual I/O cache-related information.

Note
If extended file cache (XFC) is enabled, the CLUE VCC command is disabled.

Format
CLUE VCC [/qualifier[,...]]

Parameters
None.

Qualifiers

/CACHE
Decodes and displays the cache lines that are used to correlate the file virtual block numbers (VBNs) with the memory used for caching. Note that the cache itself is not dumped in a selective dump. Use of this qualifier with a selective dump produces the following message:

%CLUE-I-VCCNOCAC, Cache space not dumped because DUMPSTYLE is selective

/LIMBO
Walks through the limbo queue (LRU order) and displays information for the cached file header control blocks (FCBs).

/STATISTIC
Displays statistical and performance information related to the virtual I/O cache.

/VOLUME
Decodes and displays the cache volume control blocks (CVCB).
Examples

1. **SDA> CLUE VCC/STATISTIC**

Virtual I/O Cache Statistics:

```
Cache State: pak,on,img,data,enabled
Cache Flags: on,protocol_only
Cache Data Area: 80855200
```

- Total Size (pages): 400
- Total Size (MBytes): 3.1 MB
- Free Size (pages): 0
- Free Size (MBytes): 0.0 MB
- Read I/O Count: 34243
- Read I/O Bypassing Cache: 3149
- Read Hit Count: 15910
- Read Hit Rate: 46.4%
- Write I/O Count: 4040
- Write I/O Bypassing Cache: 856
- I/Opost PID Action Rtns: 40829
- I/Opost Physical I/O Count: 28
- I/Opost Virtual I/O Count: 0
- Cache Line LRU Time: 00001B6E
- Limbo LRU Queue: 80B11220 80B11620
- Oldest Cache Line Time: 00001B6E
- Cache VCB Queue: 8094DE80 809A0000
- Oldest Limbo Queue Time: 00001B6F
- System Uptime: 00001BB0

2. **SDA> CLUE VCC/VOLUME**

Virtual I/O Cache - Cache VCB Queue:

```
Cache VCB  Real VCB  LockID  IRP Queue  CID  LKS  Ocmt  State
--------  --------  -------  --------  ----  ----  ----  ------
8094DEB0  80A7E440  020007B2  8094DEBC  8094DEBC  0000  0001  0002  on
8093FEC0  809F7CC0  0100022D  809F3FCC  809F3FCC  0000  0001  0002  on
809D2400  809F7A40  01000227  809D027C  809D027C  0000  0001  0002  on
80978BB0  809F6C00  01000221  80978BBC  80978BBC  0000  0001  0002  on
809AA000  809A9780  01000005  809AA3C  809AA03C  0007  0001  0002  on
```

3. **SDA> CLUE VCC/LIMBO**

Virtual I/O Cache - Limbo Queue:

```
CFCB  CVCB  FCB  CFCB  IOerrors  FID  (hex)
-----  -----  -----  -----  --------  ----  -----
80A97DC0  809AA00  80A45100  000000200  000000000  0076B,0001,00
80A4E440  809AA000  809CD040  000002000  000000000  00767,0001,00
80A63640  809AA000  809FAE80  000002000  000000000  00138,0001,00
80A2540  80978BB0  80A48140  000002000  000000000  00A5,0014,00
80A45600  809AA000  80A3AC00  000002000  000000000  00C50,0001,00
80A85C0  809AA000  809FA140  000002000  000000000  00C51,0001,00
80A69800  809AA000  809FBA00  000002000  000000000  00C52,0001,00
80951000  809AA000  80A3F140  000002000  000000000  00C53,0001,00
80A3E580  809AA000  80A11A40  000002000  000000000  00C54,0001,00
80A67800  809AA000  80978F00  000002000  000000000  00C55,0001,00
809D30C0  809AA000  809F4CC0  000002000  000000000  00C56,0001,00
809D4BB0  809AA000  8093E540  000002000  000000000  00C57,0001,00
```

[......]

- 80A81600  809AA000  809B2CC0  000002000  000000000  00C5D,0001,00
- 80A3EFC0  809AA000  80A2DECO  000002000  000000000  007EA,000A,00
- 80A98AC0  809AA000  8093C640  000002000  000000000  00C63,0001,00

```
### Virtual I/O Cache - Cache Lines:

<table>
<thead>
<tr>
<th>CL</th>
<th>VA</th>
<th>CVCB</th>
<th>CFCH</th>
<th>FCB</th>
<th>Status</th>
<th>IOerrors</th>
<th>FID (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>82B11200</td>
<td>82880000</td>
<td>80900240</td>
<td>809D7000</td>
<td>80A01100</td>
<td>000000200</td>
<td>00000000</td>
<td>(006E, 0003, 00)</td>
</tr>
<tr>
<td>82B15740</td>
<td>82AAA0000</td>
<td>809AA0000</td>
<td>80A07A00</td>
<td>80A24240</td>
<td>000000000</td>
<td>00000000</td>
<td>(0765, 0001, 00)</td>
</tr>
<tr>
<td>82B14EC0</td>
<td>82A660000</td>
<td>809AA0000</td>
<td>80A45600</td>
<td>80A3AC00</td>
<td>000000200</td>
<td>00000000</td>
<td>(0C50, 0001, 00)</td>
</tr>
<tr>
<td>82B12640</td>
<td>82922000</td>
<td>80900240</td>
<td>809D7000</td>
<td>80A01100</td>
<td>000000200</td>
<td>00000000</td>
<td>(006E, 0003, 00)</td>
</tr>
<tr>
<td>82B123C0</td>
<td>8290E0000</td>
<td>809AA0000</td>
<td>80A45600</td>
<td>80A3AC00</td>
<td>000000200</td>
<td>00000000</td>
<td>(0C50, 0001, 00)</td>
</tr>
<tr>
<td>82B13380</td>
<td>8298C0000</td>
<td>80900240</td>
<td>809D7000</td>
<td>80A01100</td>
<td>000000200</td>
<td>00000000</td>
<td>(006E, 0003, 00)</td>
</tr>
<tr>
<td>82B154A0</td>
<td>82AC2000</td>
<td>809AA0000</td>
<td>80A45600</td>
<td>80A3AC00</td>
<td>000000200</td>
<td>00000000</td>
<td>(0C50, 0001, 00)</td>
</tr>
<tr>
<td>82B15F40</td>
<td>82AEA0000</td>
<td>80900240</td>
<td>809D7000</td>
<td>80A01100</td>
<td>000000200</td>
<td>00000000</td>
<td>(006E, 0003, 00)</td>
</tr>
<tr>
<td>82B13AC0</td>
<td>82946000</td>
<td>80900240</td>
<td>809D7000</td>
<td>80A01100</td>
<td>000000200</td>
<td>00000000</td>
<td>(006E, 0003, 00)</td>
</tr>
<tr>
<td>82B12900</td>
<td>82938000</td>
<td>80900240</td>
<td>809D7000</td>
<td>80A01100</td>
<td>000000200</td>
<td>00000000</td>
<td>(006E, 0003, 00)</td>
</tr>
<tr>
<td>82B10280</td>
<td>82804000</td>
<td>809AA0000</td>
<td>80A45600</td>
<td>80A3AC00</td>
<td>000000200</td>
<td>00000000</td>
<td>(0C50, 0001, 00)</td>
</tr>
<tr>
<td>82B122C0</td>
<td>82906000</td>
<td>809AA0000</td>
<td>80A480000</td>
<td>000000000</td>
<td>00000000</td>
<td>(0164, 0001, 00)</td>
<td></td>
</tr>
<tr>
<td>82B14700</td>
<td>82A280000</td>
<td>809AA0000</td>
<td>809FE0C0</td>
<td>809FEDC0</td>
<td>00000004</td>
<td>00000000</td>
<td>(07B8, 0001, 00)</td>
</tr>
<tr>
<td>82B11A40</td>
<td>82890000</td>
<td>809AA0000</td>
<td>80A113C0</td>
<td>80A11840</td>
<td>00000000</td>
<td>00000000</td>
<td>(00AF, 0001, 00)</td>
</tr>
</tbody>
</table>

[...]
CLUE XQP

Displays XQP-related information.

Format

CLUE XQP [/qualifier[,...]]

Parameters

None.

Qualifiers

/ACTIVE [/FULL]
Displays all active XQP processes.

/AQB
Displays any current I/O request packets (IRPs) waiting at the interlocked queue.

/BFRD=index
Displays the buffer descriptor (BFRD) referenced by the index specified. The index is identical to the hash value.

/BFRL=index
Displays the buffer lock block descriptor (BFRL) referenced by the index specified. The index is identical to the hash value.

/BUFFER=(n,m) [/FULL]
Displays the BFRDs for a given pool. Specify either 0, 1, 2 or 3, or a combination of these in the parameter list.

/CACHE_HEADER
Displays the block buffer cache header.

/FCB=address [/FULL]
Displays all file header control blocks (FCBs) with a nonzero DIRINDX for a given volume. If no address is specified, the current volume of the current process is used.

The address specified can also be either a valid volume control block (VCB), unit control block (UCB), or window control block (WCB) address.

/FILE=address
Decodes and displays file header (FCB), window (WCB), and cache information for a given file. The file can be identified by either its FCB or WCB address.

/GLOBAL
Displays the global XQP area for a given process.

/LBN_HASH=lbn
Calculates and displays the hash value for a given logical block number (LBN).
SDA CLUE Extension Commands

CLUE XQP

/LIMBO
Searches through the limbo queue and displays FCB information from available, but unused file headers.

/LOCK=lockbasis
Displays all file system serialization, arbitration, and cache locks found for the specified lockbasis.

/THREAD=n
Displays the XQP thread area for a given process. The specified thread number is checked for validity. If no thread number is specified, the current thread is displayed. If no current thread, but only one single thread is in use, then that thread is displayed. If more than one thread exists or an invalid thread number is specified, then a list of currently used threads is displayed.

/VALIDATE=(n,m)
Performs certain validation checks on the block buffer cache to detect corruption. Specify 1, 2, 3, 4, or a combination of these in the parameter list. If an inconsistency is found, a minimal error message is displayed. If you add the /FULL qualifier, additional information is displayed.

Description
The CLUE XQP command displays XQP information. XQP is part of the I/O subsystem.

Examples

1. SDA> CLUE XQP/CACHE_HEADER
   Block Buffer Cache Header:
   ---------------------------------------------
   Cache_Header 8437DF90  BFRcnt  000005D2  FreeBFRL  843916A0
   Bufbase 8439B400  BFRDbase  8437E080  BFRLbase  8438F7E0
   Bufsize 000BA400  LBNHashtbl  84398390  BFRLHashtbl  84399BC8
   Realsize 00D78A0  LBNHashcnt  0000060E  BFRLHashcnt  0000060E
   Pool #0 #1 #2 #3
   Pool_LRU 8437E5C0 84385F40 84387E90 8438EEB0
   8437F400 84385D60 8438AC80 8438EE20
   Pool_WAITQ 8437DFE0 8437DFE8 8437DFF0 8437DFF8
   8437DFE0 8437DFE8 8437DFF0 8437DFF8
   Waitcnt 00000000 00000000 00000000 00000000
   Poolavail 00000094 00000252 00000251 00000094
   Poolcnt 00000095 00000254 00000254 00000095
   AmbigQFL 00000000  Process_Hits  00000000  Cache_Serial  00000000
   AmbigQL 00000000  Valid_Hits  00000000  Cache_Stalls  00000000
   Disk_Reads 00000000  Invalid_Hits  00000000  Buffer_Stalls  00000000
   Disk_Writes 00000000  Misses  00000000

   The SDA command CLUE XQP/CACHE_HEADER displays the block buffer cache header.

2. SDA> CLUE XQP/VALIDATE=(1,4)
   Searching BFRD Array for possible Corruption...
   Searching Lock Basis Hashtable for possible Corruption...

   In this example, executing the CLUE XQP/VALIDATE=1,4 command indicated that no corruption was detected in either the BFRD Array or the Lock Basis Hashtable.
This chapter presents an overview of the SDA Spinlock Tracing Utility commands, and describes the SDA Spinlock Tracing commands.

6.1 Overview of the SDA Spinlock Tracing Utility

To synchronize access to data structures, the OpenVMS operating system uses a set of static spinlocks, such as IOLOCK8 and SCHED. The operating system acquires a spinlock to synchronize data, and at the end of the critical code path the spinlock is then released. If a CPU attempts to acquire a spinlock while another CPU is holding it, the CPU attempting to acquire the spinlock has to spin, waiting until the spinlock is released. Any lost CPU cycles within such a spinwait loop are charged as MPsynch time.

By using the MONITOR utility, you can monitor the time in process modes, for example, with the command $ MONITOR MODES. A high rate of MP synchronization indicates contention for spinlocks. However, until the implementation of the Spinlock Tracing utility, there was no way to tell which spinlock was heavily used, and who was acquiring and releasing the contended spinlocks. The Spinlock Tracing utility allows a characterization of spinlock usage. It can also collect performance data for a given spinlock on a per-CPU basis.

This tracing ability is built into the system synchronization execlet, which contains the spinlock code, and can be enabled or disabled while the system is running. There is no need to reboot the system to load a separate debug image. The images that provide spinlock tracing functionality are as follows:

- SYS$LOADABLE_IMAGES:SPL$DEBUG.EXE
- SYS$SHARE:SPL$SDA.EXE

The SDA> prompt provides the command interface. From this command interface, you can load and unload the spinlock debug execlet using SPL LOAD and SPL UNLOAD, and start, stop and display spinlock trace data. This allows you to collect spinlock data for a given period of time without system interruption. Once information is collected, the trace buffer can be deallocated and the execlet can be unloaded to free up system resources. The spinlock trace buffer is allocated from S2 space and pages are taken from the freelist.

Should the system crash while spinlock tracing is enabled, the trace buffer is dumped into the system dump file, and it can later be analyzed using the spinlock trace utility. This is very useful in tracking down CPUSPINWAIT bugcheck problems.

Note that by enabling spinlock tracing, there is a performance impact. The amount of the impact depends on the amount of spinlock usage.
6.1 Overview of the SDA Spinlock Tracing Utility

Note

The Spinlock Tracing utility is still under development. The command format, displays, and suggested approach to spinlock analysis are all subject to change.

6.2 How to Use the SDA Spinlock Tracing Utility

The following steps will enable you to collect spinlock statistics using the Spinlock Tracing Utility.

1. Load the Spinlock Tracing Utility execut.
   ```
   SDA> SPL LOAD
   ```

2. Allocate a trace buffer and start tracing.
   ```
   SDA> START TRACE
   ```

3. Wait a few seconds to allow some tracing to be done, then find out which spinlocks are incurring the most acquisitions and the most spinwaits.
   ```
   SDA> SHOW TRACE/SUMMARY
   ```

   For example, you might see contention for the SCHED and IOLOCK8 spinlocks (a high acquisition count, with a significant proportion of the acquisitions being forced to wait).

4. Look to see if the spinlocks with a high proportion of spinwaits caused a significant delay in the acquisition of the spinlock. You must now collect more detailed statistics on a specific spinlock.
   ```
   SDA> SPL START COLLECT/SPINLOCK=SCHED
   ```

   This command accumulates additional data for the specified spinlock. As long as tracing is not stopped, collection will continue to accumulate spinlock-specific data from the trace buffer.

5. Display the additional data collected for the specified spinlock.
   ```
   SDA> SHOW COLLECT
   ```

   This display includes the average hold time of the spinlock and the average spinwait time while acquiring the spinlock.

6. Repeat steps 4 and 5 for each spinlock that has contention. A START COLLECT cancels the previous collection.

7. Disable spinlock tracing when you have collected all the needed spinlock statistics and release all the memory used by the Spinlock Tracing utility with the following commands.
   ```
   SDA> SPL STOP COLLECT
   SDA> SPL STOP TRACE
   SDA> SPL UNLOAD
   ```
6.3 Example Command Procedure for Collection of Spinlock Statistics

The following example shows a command procedure that can be used for gathering spinlock statistics:

```bash
$ analyze/system
  spl load
  spl start trace/buffer=1000
  spawn wait 00:00:15
  spl stop trace
  read/executive/nolog
  set output spl_trace.lisspl
  spl show trace/summary
  spl start collect/spin=sched
  spawn wait 00:00:05
  spl show collect
  spl start collect/spin=iolock8
  spawn wait 00:00:05
  spl show collect
  spl start collect/spin=lockmgr
  spawn wait 00:00:05
  spl show collect
  spl start collect/spin=mmg
  spawn wait 00:00:05
  spl show collect
  spl start collect/spin=timer
  spawn wait 00:00:05
  spl show collect
  spl start collect/spin=mailbox
  spawn wait 00:00:05
  spl show collect
  spl start collect/spin=perfmon
  spawn wait 00:00:05
  spl show collect
  spl stop collect
  spl unload
  exit

$ exit
```

A more comprehensive procedure is provided as SYS$EXAMPLES:SPL.COM.

6.4 Listing of SDA Spinlock Tracing Commands

The following is a list of the spinlock tracing commands:

- SPL LOAD
- SPL SHOW COLLECT
- SPL SHOW TRACE
- SPL START COLLECT
- SPL START TRACE
- SPL STOP COLLECT
- SPL STOP TRACE
- SPL UNLOAD
SPL LOAD

Loads the SPL$DEBUG execlet. This must be done prior to starting spinlock tracing.

Format

SPL LOAD

Parameters

None.

Qualifiers

None.

Description

The SPL LOAD command loads the SPL$DEBUG execlet, which contains the tracing routines.

Example

SDA> SPL LOAD
SPL$DEBUG load status = 00000001
SPL SHOW COLLECT

Displays the collected spinlock data.

Format

SPL SHOW COLLECT

Parameters

None.

Qualifiers

None.

Description

The SPL SHOW COLLECT command displays the collected spinlock data. It displays first a summary on a per-CPU basis, followed by the callers of the specific spinlock. This second list is sorted by the top consumers of the spinlock (in system cycles). These displays show average spinlock hold and spinlock wait time in system cycles.

Example

SDA> SPL SHOW COLLECT
Spinlock Trace Information for SCHED:
-------------------------------------
<table>
<thead>
<tr>
<th>CPU Id</th>
<th>Spinhold Time</th>
<th>Total Count</th>
<th>Average Hold</th>
<th>Spinwait Time</th>
<th>Total Count</th>
<th>Average Spin</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>250691522</td>
<td>120295</td>
<td>2083</td>
<td>115690429</td>
<td>18903</td>
<td>6120</td>
</tr>
<tr>
<td>03</td>
<td>252806297</td>
<td>143816</td>
<td>1757</td>
<td>110733404</td>
<td>18638</td>
<td>5941</td>
</tr>
<tr>
<td>04</td>
<td>10149145</td>
<td>47524</td>
<td>2135</td>
<td>222554324</td>
<td>13663</td>
<td>16288</td>
</tr>
<tr>
<td>05</td>
<td>91030899</td>
<td>41179</td>
<td>2210</td>
<td>222879724</td>
<td>13498</td>
<td>16512</td>
</tr>
<tr>
<td>06</td>
<td>84989539</td>
<td>40070</td>
<td>2121</td>
<td>217136237</td>
<td>12774</td>
<td>16598</td>
</tr>
<tr>
<td>07</td>
<td>81678817</td>
<td>36754</td>
<td>2222</td>
<td>219025091</td>
<td>12865</td>
<td>17039</td>
</tr>
<tr>
<td>08</td>
<td>160358528</td>
<td>67101</td>
<td>2389</td>
<td>220799983</td>
<td>14681</td>
<td>15039</td>
</tr>
<tr>
<td>09</td>
<td>93886644</td>
<td>43778</td>
<td>2144</td>
<td>223951595</td>
<td>13476</td>
<td>16618</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1116931391</td>
<td>540517</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Spinlock Trace Information for SCHED: (8-SEP-2000 07:55:15.44, 1.9 nsec, 523 MHz)
-----------------------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Callers PC</th>
<th>Total Cycles</th>
<th>Tot Count</th>
<th>Maximum</th>
<th>Minimum</th>
<th>Average</th>
<th>Spinwaits</th>
<th>Ave Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>80111650</td>
<td>SCHR$CALC_CPU LOAD_C+00500</td>
<td>338642620</td>
<td>106300</td>
<td>53627</td>
<td>687</td>
<td>78681</td>
<td>15742</td>
</tr>
<tr>
<td>8004B34C</td>
<td>SCHR$UNLOCK QUAD_C+0050C</td>
<td>102106193</td>
<td>88114</td>
<td>61818</td>
<td>217</td>
<td>6661</td>
<td>7463</td>
</tr>
<tr>
<td>8004A16A</td>
<td>SCHR$LOCKR QUAD_C+00034</td>
<td>78434764</td>
<td>61253</td>
<td>56934</td>
<td>221</td>
<td>4620</td>
<td>10878</td>
</tr>
<tr>
<td>80132754</td>
<td>SCHR$LOCK QUAD_C+00054</td>
<td>75029658</td>
<td>25968</td>
<td>54509</td>
<td>292</td>
<td>2889</td>
<td>6108</td>
</tr>
<tr>
<td>80134807</td>
<td>SCHR$POSTEF_C+00000</td>
<td>6940932</td>
<td>30384</td>
<td>56064</td>
<td>317</td>
<td>2055</td>
<td>6950</td>
</tr>
<tr>
<td>8013E04A</td>
<td>EKESYNCH LOOP_C+00458</td>
<td>50977238</td>
<td>9427</td>
<td>71237</td>
<td>1602</td>
<td>5407</td>
<td>8041</td>
</tr>
<tr>
<td>8004A3E4</td>
<td>SCHR$LOCKR QUAD_C+00034</td>
<td>47965790</td>
<td>28281</td>
<td>59602</td>
<td>298</td>
<td>1685</td>
<td>9704</td>
</tr>
<tr>
<td>8004A3E4</td>
<td>SCHR$LOCK QUAD_C+00000C</td>
<td>29557509</td>
<td>19805</td>
<td>19917</td>
<td>235</td>
<td>1485</td>
<td>1430</td>
</tr>
<tr>
<td>8012F4A4</td>
<td>PROCESS MANAGEMENT+1F4E4</td>
<td>29513273</td>
<td>4767</td>
<td>47172</td>
<td>388</td>
<td>6191</td>
<td>1721</td>
</tr>
<tr>
<td>8011A900</td>
<td>SCHR$QEND_C+00080</td>
<td>26705966</td>
<td>4258</td>
<td>52945</td>
<td>274</td>
<td>6271</td>
<td>166</td>
</tr>
</tbody>
</table>

...
SPL SHOW TRACE

Displays spinlock tracing information.

Format

SPL SHOW TRACE [/[NO]SPINLOCK=spinlock][/[NO]FORKLOCK=forklock
/[NO]ACQUIRE |/[NO]RELEASE |/[NO]WAIT
/[NO]FRKDSPTH |/[NO]FRKEND
/[SUMMARY |/CPU=n |/TOP=n]

Parameters

None.

Qualifiers

/SPINLOCK=spinlock
/NOSPINLOCK
The /SPINLOCK=n qualifier specifies the display of a specific spinlock, for example, /SPINLOCK=LCKMGR or /SPINLOCK=SCHED.
The /NOSPINLOCK qualifier specifies that no spinlock trace information be displayed. If omitted, all spinlock trace entries are decoded and displayed.

/FORKLOCK=forklock
/NOFORKLOCK
The /FORKLOCK=forklock qualifier specifies the display of a specific forklock, for example, /FORKLOCK=IOLOCK8 or /FORKLOCK=PL8.
The /NOFORKLOCK qualifier specifies that no forklock trace information be displayed. If omitted, all fork trace entries are decoded and displayed.

/ACQUIRE
/NOACQUIRE
The /ACQUIRE qualifier displays any spinlock acquisitions.
The /NOACQUIRE qualifier ignores any spinlock acquisitions.

/RELEASE
/NORELEASE
The /RELEASE qualifier displays any spinlock releases.
The /NORELEASE qualifier ignores any spinlock releases.

/WAIT
/NOWAIT
The /WAIT qualifier displays any spinwait operations.
The /NOWAIT qualifier ignores any spinwait operations.

/FRKDSPTH
/NOFRKDSPTH
The /FRKDSPTH qualifier displays all invocations of fork routines within the fork dispatcher. This is the default.
The /NOFRKDSPTH qualifier ignores all of the operations of the /FRKDSPTH qualifier.
The /FRKEND qualifier displays all returns from fork routines within the fork dispatcher. This is the default.

The /NOFRKEND qualifier ignores all operations of the /FRKEND qualifier.

/CPU=n
Specifies the display of information for a specific CPU only, for example, /CPU=5 or /CPU=PRIMARY. By default, all trace entries for all CPUs are displayed.

/SUMMARY
Steps through the entire trace buffer and displays a summary of all spinlock and forklock activity. It also displays the top ten callers.

/TOP=n
Displays a different number other than the top ten callers or fork PCs. By default, the top ten are displayed. This qualifier is only useful when you also specify the /SUMMARY qualifier.

**Description**

The SPL SHOW TRACE command displays spinlock tracing information. The latest acquired or released spinlock is displayed first, and then the trace buffer is stepped backwards in time.

By default, all trace entries will be displayed, but you can use qualifiers to select only certain entries.

Since this is not a time critical activity and a table lookup has to be done anyway to translate the SPL address to a spinlock name, commands like /SPINLOCK=(SCHED,IOLOCK8) do work. /SUMMARY will step the entire trace buffer and display a summary of all spinlock activity, along with the top-ten callers' PCs. You can use /TOP=n to display a different number of the top ranked callers.
### Callout Meaning

1. Shows timestamps that are collected as system cycle counters (SCC) and then displayed with an accuracy down to microseconds. Each CPU is incrementing its own SCC as soon as it is started, so there is some difference between different CPUs’ system cycle counters. The standard system time is incremented only every 10 Msec and as such is not exact enough. Adjusting the SCC to the specific CPU’s system time and translating it into an accurate timestamp will thus sometimes display times out of order for different CPUs. However, for the same CPU ID, the timestamps are accurate.

2. Shows the physical CPU ID of the CPU logging the trace entry.

3. Shows the address of the spinlock fork. If it is a static one, its name is displayed; otherwise, it is marked as ??????.

4. Shows the caller’s PC address that acquired or released the spinlock, or the fork PC if the trace entry is a forklock. Symbolization is attempted, so a READ/EXECUTIVE might help to display a routine name, instead of simply a module and offset.

5. Shows the EPID, which is the external PID of the process generating the trace entry. If an interrupt or fork was responsible for the entry, then a zero EPID is displayed.

6. Shows the trace operation. For a spinlock, which was acquired without going through a spinwait, there is a matching acquire/release pair of trace entries for the same CPU ID for a given spinlock. If a spinlock is held, it cannot be acquired immediately, so there is also a spinwait trace entry for this pair. The different variations of the acquire and release operations are distinguished, as are the same spinlocks if they are acquired recursively multiple times.

7. Shows the address of the trace buffer entry, in case there is a need to access the raw and undecoded trace data.

<table>
<thead>
<tr>
<th>Callout</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shows timestamps that are collected as system cycle counters (SCC) and then displayed with an accuracy down to microseconds. Each CPU is incrementing its own SCC as soon as it is started, so there is some difference between different CPUs’ system cycle counters. The standard system time is incremented only every 10 Msec and as such is not exact enough. Adjusting the SCC to the specific CPU’s system time and translating it into an accurate timestamp will thus sometimes display times out of order for different CPUs. However, for the same CPU ID, the timestamps are accurate.</td>
</tr>
<tr>
<td>2</td>
<td>Shows the physical CPU ID of the CPU logging the trace entry.</td>
</tr>
<tr>
<td>3</td>
<td>Shows the address of the spinlock fork. If it is a static one, its name is displayed; otherwise, it is marked as ??????.</td>
</tr>
<tr>
<td>4</td>
<td>Shows the caller’s PC address that acquired or released the spinlock, or the fork PC if the trace entry is a forklock. Symbolization is attempted, so a READ/EXECUTIVE might help to display a routine name, instead of simply a module and offset.</td>
</tr>
<tr>
<td>5</td>
<td>Shows the EPID, which is the external PID of the process generating the trace entry. If an interrupt or fork was responsible for the entry, then a zero EPID is displayed.</td>
</tr>
<tr>
<td>6</td>
<td>Shows the trace operation. For a spinlock, which was acquired without going through a spinwait, there is a matching acquire/release pair of trace entries for the same CPU ID for a given spinlock. If a spinlock is held, it cannot be acquired immediately, so there is also a spinwait trace entry for this pair. The different variations of the acquire and release operations are distinguished, as are the same spinlocks if they are acquired recursively multiple times.</td>
</tr>
<tr>
<td>7</td>
<td>Shows the address of the trace buffer entry, in case there is a need to access the raw and undecoded trace data.</td>
</tr>
</tbody>
</table>
2. SDA SPL SHOW TRACE /SUMMARY

Spinlock Trace Information: (at 8-SEP-2000 07:55:02.66, trace time 00:02:04.873664)

<table>
<thead>
<tr>
<th>Spinlock</th>
<th>Total Events</th>
<th>Acquire Events</th>
<th>Release Events</th>
<th>Acquire Own</th>
<th>Acquire Nospin</th>
<th>Acquire Inuse</th>
<th>Spinwaits</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEGA</td>
<td>248</td>
<td>124</td>
<td>124</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HWCLK</td>
<td>255316</td>
<td>127658</td>
<td>127658</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INVALIDATE</td>
<td>1588</td>
<td>794</td>
<td>794</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>POOL</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MAILBOX</td>
<td>640</td>
<td>162</td>
<td>327</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>SCHED</td>
<td>213283</td>
<td>90652</td>
<td>90690</td>
<td>85</td>
<td>0</td>
<td>0</td>
<td>31856</td>
</tr>
<tr>
<td>MMG</td>
<td>282723</td>
<td>104646</td>
<td>104647</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>73430</td>
</tr>
<tr>
<td>TIMER</td>
<td>45138</td>
<td>22567</td>
<td>22567</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>TX_SYNCH</td>
<td>4119</td>
<td>2059</td>
<td>2059</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IOLocks</td>
<td>30374</td>
<td>15140</td>
<td>15140</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>???</td>
<td>42567</td>
<td>21107</td>
<td>21269</td>
<td>165</td>
<td>0</td>
<td>0</td>
<td>29</td>
</tr>
</tbody>
</table>

Spinlock Trace Information:

<table>
<thead>
<tr>
<th>Spinlock</th>
<th>Total Count Aquire</th>
<th>Release</th>
<th>Wait</th>
<th>Own</th>
<th>Callers PC</th>
<th>Module</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHED</td>
<td>56404</td>
<td>30142</td>
<td>0</td>
<td>26262</td>
<td>0</td>
<td>SCHCALC_CPU_LOAD_C+00580</td>
<td>PROCESS_MANAGEMENT</td>
</tr>
<tr>
<td>SCHED</td>
<td>30146</td>
<td>30146</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCHCALC_CPU_LOAD_C+00300</td>
<td>PROCESS_MANAGEMENT</td>
</tr>
<tr>
<td>SCHED</td>
<td>11452</td>
<td>10988</td>
<td>0</td>
<td>464</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>SCHED</td>
<td>10988</td>
<td>10988</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+00124</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>SCHED</td>
<td>6220</td>
<td>6220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$LOCKW_QUAD_C+00034</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>SCHED</td>
<td>6159</td>
<td>4527</td>
<td>0</td>
<td>1632</td>
<td>0</td>
<td>SCH$LOCKR_QUAD_C+0007C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>SCHED</td>
<td>5301</td>
<td>4794</td>
<td>0</td>
<td>507</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>SCHED</td>
<td>5112</td>
<td>4706</td>
<td>0</td>
<td>383</td>
<td>23</td>
<td>SCH$CALC_CPU_LOAD_C+00580</td>
<td>PROCESS_MANAGEMENT</td>
</tr>
<tr>
<td>MMG</td>
<td>43675</td>
<td>28296</td>
<td>0</td>
<td>15379</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>42554</td>
<td>21603</td>
<td>0</td>
<td>20951</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>35460</td>
<td>19148</td>
<td>0</td>
<td>16312</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>26731</td>
<td>26731</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>21604</td>
<td>21604</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>21600</td>
<td>21600</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>16559</td>
<td>16559</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>3443</td>
<td>3432</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
<tr>
<td>MMG</td>
<td>3432</td>
<td>3432</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SCH$UNLOCK_QUAD_C+0005C</td>
<td>SYSTEM_PRIMITIVES_MIN</td>
</tr>
</tbody>
</table>

Callout Meaning

8 Shows the summary information by stepping through the whole trace buffer, and displaying a single line of information for each spinlock. If the number of spinwaits compared to the number of acquisitions is very high, then a spinlock is a candidate for high contention.

9 For each spinlock in the summary display, the top ten callers’ PCs are displayed along with the number of spinlock acquisitions and releases, as well as spinwait counts and the number of multiple acquisitions of the same spinlock.
Forklock Trace Information:  (at 8-SEP-2000 07:55:02.66, trace time 00:02:04.873664)

<table>
<thead>
<tr>
<th>Forklock</th>
<th>Total Events</th>
<th>CPU IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPL 08</td>
<td>3757</td>
<td>2  0  0  0  0  0  0  0  0</td>
</tr>
<tr>
<td>TIMER</td>
<td>7047</td>
<td>7047   0  0  0  0  0  0  0  0</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>12747</td>
<td>12180  357 58 27 35 16 35 39</td>
</tr>
<tr>
<td>LCMGR</td>
<td>179</td>
<td>20  97  10  7  9  9  21  6</td>
</tr>
<tr>
<td>QUEUEST</td>
<td>326</td>
<td>33  188 22 14 15 13 35 6</td>
</tr>
</tbody>
</table>

---

Forklock Trace Information:

<table>
<thead>
<tr>
<th>Forklock</th>
<th>Tot Count</th>
<th>Tot Cycles</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Fork PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPL 08</td>
<td>2913</td>
<td>00:00:00.049783</td>
<td>8947</td>
<td>1161</td>
<td>231717</td>
<td>803C5890 SYSSPCADVR+01890</td>
</tr>
<tr>
<td>IPL 08</td>
<td>844</td>
<td>00:00:00.017516</td>
<td>10865</td>
<td>6053</td>
<td>100257</td>
<td>80020300 LANSFOR_SD_RELEASESDU_C</td>
</tr>
<tr>
<td>TIMER</td>
<td>7047</td>
<td>00:00:00.109392</td>
<td>10865</td>
<td>6053</td>
<td>100257</td>
<td>8004DBF0 EXE$SWTIMINT_C+00180</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>8130</td>
<td>00:00:00.071646</td>
<td>4613</td>
<td>1947</td>
<td>71974</td>
<td>80590200 SYSSPEDRIVER+11B98</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>3783</td>
<td>00:00:00.034208</td>
<td>4734</td>
<td>1583</td>
<td>9160</td>
<td>805B9B98 TCPIPINTERNET SERVICES+1FB60</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>255</td>
<td>00:00:00.003208</td>
<td>1769</td>
<td>6284</td>
<td>2697</td>
<td>8068B9F8 SYSSTDADVR+03F30</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>27</td>
<td>00:00:00.000105</td>
<td>1769</td>
<td>6284</td>
<td>2697</td>
<td>8068B9F8 SYSSTDADVR+03F30</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>16</td>
<td>00:00:00.000394</td>
<td>2906</td>
<td>1352</td>
<td>6536</td>
<td>80083300 SMP$CPU_SWITCH_C+00300</td>
</tr>
<tr>
<td>IOLOCK8</td>
<td>5</td>
<td>00:00:00.000009</td>
<td>2906</td>
<td>1352</td>
<td>6536</td>
<td>80083300 SMP$CPU_SWITCH_C+00300</td>
</tr>
<tr>
<td>LCMGR</td>
<td>87</td>
<td>00:00:00.001728</td>
<td>8730</td>
<td>373</td>
<td>19658</td>
<td>800E75C0 NETDRIVER+005D0</td>
</tr>
<tr>
<td>LCMGR</td>
<td>5</td>
<td>00:00:00.000000</td>
<td>8730</td>
<td>373</td>
<td>19658</td>
<td>800E75C0 NETDRIVER+005D0</td>
</tr>
<tr>
<td>QUEUEST</td>
<td>71</td>
<td>00:00:00.000147</td>
<td>52705</td>
<td>17451</td>
<td>69920</td>
<td>800CA90 SYSSPCACHE+A1990</td>
</tr>
<tr>
<td>QUEUEST</td>
<td>71</td>
<td>00:00:00.000103</td>
<td>8075</td>
<td>3024</td>
<td>10700</td>
<td>800CA90 SYSSPCACHE+A3010</td>
</tr>
<tr>
<td>QUEUEST</td>
<td>71</td>
<td>00:00:00.000002</td>
<td>2009</td>
<td>4125</td>
<td>4164</td>
<td>80208B10 CACHETM_SERVICE+004B0</td>
</tr>
<tr>
<td>QUEUEST</td>
<td>14</td>
<td>00:00:00.000000</td>
<td>63850</td>
<td>21244</td>
<td>183880</td>
<td>80208B30 SYSSPCACHE+1DE30</td>
</tr>
<tr>
<td>QUEUEST</td>
<td>1</td>
<td>00:00:00.000000</td>
<td>5184</td>
<td>5184</td>
<td>5184</td>
<td>8007EE0 IOC_STDSPFREE UCB_C+005050</td>
</tr>
</tbody>
</table>

**Callout Meaning**

10 The forklock summary displays the number of fork operations on a specific CPU for each forklock. For each forklock, the top ten fork PC addresses are displayed, along with the minimum, maximum and average duration of the fork operation in system cycles. The total amount of time spent in a given fork routine is displayed in a time format accurate to microseconds.
**SPL START COLLECT**

Starts to collect spinlock information a longer period of time than will fit into the trace buffer.

**Format**

```
SPL START COLLECT   [/SPINLOCK=spinlock] [/ADDRESS=n]
```

**Parameters**

None.

**Qualifiers**

- `/SPINLOCK=spinlock`
  Specifies the tracing of a specific spinlock, for example, `/SPINLOCK=LCKMGR` or `/SPINLOCK=SCHED`.

- `/ADDRESS=n`
  Specifies the tracing of a specific spinlock by address.

**Description**

The SPL START COLLECT command starts a collection of spinlock information for a longer period of time than will fit into the trace buffer. You need to enable spinlock tracing before a spinlock collection can be started. On a system with heavy activity, the trace buffer typically can only hold a relatively small time window of spinlock information. In order to collect spinlock information over a longer time period, a collection can be started. The collection tries to catch up with the running trace index and save the spinlock information into a balanced tree within the virtual address space of the process performing the spinlock collection. Either use the name of a static spinlock, or supply the address of a dynamic spinlock, for which information should be gathered.

The trace entries are kept in the trace buffer, which is allocated from S2 space, hence there is no disruption, if tracing is started from within SDA and then the user exits from SDA. However, for the longer period data collection, the information is kept in process-specific memory, thus a user needs to stay within SDA; otherwise the data collection is automatically terminated by SDA's image rundown. You can collect data for two or more spinlocks simultaneously, by using a separate process for each collection.

**Examples**

```
SDA> SPL START COLLECT
Use /SPINLOCK=name or /ADDRESS=n to specify which spinlock info needs to be collected...

This example shows that you need to supply either a spinlock name of a static spinlock, or the address of a dynamic spinlock, if you want to collect information over a long period of time.

SDA> SPL START COLLECT/SPINLOCK=LCKMGR

This example shows the command line to start to collect information on the usage of the LCKMGR spinlock.
```
SPL START TRACE

Enables spinlock tracing.

Format

SPL START TRACE

$\begin{align*}
| & /[\text{NO]}\text{SPINLOCK}=\text{spinlock} | /[\text{NO]}\text{FORKLOCK}=\text{forklock} \\
| & /\text{BUFFER}=\text{pages} | /[\text{NO]}\text{ACQUIRE} \\
| & /[\text{NO]}\text{RELEASE} | /[\text{NO]}\text{WAIT} | /[\text{NO]}\text{FRKDSPTH} \\
| & /[\text{NO]}\text{FRKEND} | /\text{CPU}=n \\
\end{align*}$

Parameters

None.

Qualifiers

/SPINLOCK=spinlock
/NOSPINLOCK
The /SPINLOCK=spinlock qualifier specifies the tracing of a specific spinlock, for example, /SPINLOCK=LCKMGR or /SPINLOCK=SCHED.

The /NOSPINLOCK qualifier disables spinlock tracing and does not collect any spinlock data. If omitted, all spinlocks are traced.

/FORKLOCK=forklock
/NOFORKLOCK
The /FORKLOCK=forklock qualifier specifies the tracing of a specific forklock, for example, /FORKLOCK=IOLOCK8 or /FORKLOCK=IPL8.

The /NOFORKLOCK qualifier disables forklock tracing and does not collect any forklock data. If omitted, all forks are traced.

/BUFFER=pages
Specifies the size of the trace buffer (in Alpha page units). It defaults to 128 pages, which is equivalent to 1MB, if omitted.

/ACQUIRE
/NOACQUIRE
The /ACQUIRE qualifier traces any spinlock acquisitions. This is the default.

The /NOACQUIRE qualifier ignores any spinlock acquisitions.

/RELEASE
/NORELEASE
The /RELEASE qualifier traces any spinlock releases. This is the default.

The /NORELEASE qualifier ignores any spinlock releases.

/WAIT
/NOWAIT
The /WAIT qualifier traces any spinwait operations. This is the default.

The /NOWAIT qualifier ignores any spinwait operations.
/FRKDSPTH
/NOFRKDSPTH
The /FRKDSPTH qualifier traces all invocations of fork routines within the fork dispatcher. This is the default.

The /NOFRKDSPTH qualifier ignores all of the /FRKDSPTH operations.

/FRKEND
/NOFRKEND
The /FRKEND qualifier traces all returns from fork routines within the fork dispatcher. This is the default.

The /NOFRKEND qualifier ignores all of the operations of the /FRKEND qualifier.

/CPU=n
Specifies the tracing of a specific CPU only, for example, /CPU=5 or /CPU=PRIMARY. By default, all CPUs are traced.

Description
The SPL START TRACE command enables spinlock and fork tracing. By default all spinlocks and forks are traced and a 128 page (1MByte) trace buffer is allocated and used as a ring buffer.

Examples
1. SDA> SPL START TRACE/BUFFER=1000
   Tracing started... (Spinlock = 00000000, Forklock = 00000000)
   This example shows how to enable a tracing for all spinlock and forklock operations into a 8 MByte trace buffer.

2. SDA> SPL START TRACE/CPU=PRIMARY/SPINLOCK=SCHED /NOFORKLOCK
   Tracing started... (Spinlock = 810AF600, Forklock = 00000000)
   This example shows how to trace only SCHED spinlock operations on the primary CPU.

3. SDA> SPL START TRACE /NOSPINLOCK /FORKLOCK=IPL8
   Tracing started... (Spinlock = 00000000, Forklock = 863A4C00)
   This example shows how to trace only fork operations to IPL8.
SPL STOP COLLECT

Stops the spinlock collection, but does not stop spinlock tracing.

Format

SPL STOP COLLECT

Parameters

None.

Qualifiers

None.

Description

The SPL STOP COLLECT command stops the data collection, but does not affect tracing. This allows the user to start another collection for a different spinlock during the same trace run.

Example

SDA> SPL STOP COLLECT
SPL STOP TRACE

Disables spinlock tracing, but it does not deallocate the trace buffer.

Format

SPL STOP TRACE

Parameters

None.

Qualifiers

None.

Description

The SPL STOP TRACE command stops tracing, but leaves the trace buffer allocated for further analysis.

Example

SDA> SPL STOP TRACE
Tracing stopped...
Unloads the SPL$DEBUG execlet and performs cleanup. Tracing is automatically disabled and the trace buffer deallocated.

Format
SPL UNLOAD

Parameters
None.

Qualifiers
None.

Description
The SPL UNLOAD command disables the tracing or collection functionality with a delay to a state of quiescence. This ensures that all pending trace operations in progress have finished before the trace buffer is deallocated. Finally the SPL UNLOAD command unloads the SPL$DEBUG execlet.

Example
SDA> SPL UNLOAD
SPL$DEBUG unload status = 00000001
This chapter describes how to write, debug, and invoke an SDA Extension. This chapter also describes the routines available to an SDA Extension.

### 7.1 Introduction

When analysis of a dump file or a running system requires intimate knowledge of data structures that are not known to the System Dump Analyzer, the functionality of SDA can be extended by the addition of new commands into which the necessary knowledge has been built. Note that in this description, whenever a reference is made to accessing a dump file (ANALYZE/CRASH_DUMP), this also includes accessing memory in the running system (ANALYZE/SYSTEM).

For example, a user-written device driver allocates nonpaged pool and records additional data about the device there (logging different types of I/O, perhaps), and a pointer to the new structure is saved in the device-specific extension of the UCB. After a system crash, the only way to look at the data from SDA is to do the following:

- Invoke the SDA command DEFINE to define a new symbol (for example, UCB$L_FOOBAR) whose value is the offset in the UCB of the pointer to the new structure.
- Invoke the SDA commands "SHOW DEVICE <device>" and "FORMAT UCB" to obtain the address of the nonpaged pool structure.
- Invoke the SDA command "EXAMINE <address><length>" to display the contents of the data in the new nonpaged pool structure as a series of hexadecimal longwords.
- Decode manually the contents of the data structure from this hexadecimal dump.

An SDA extension that knows the layout of the nonpaged pool structure, and where to find the pointer to it in the UCB, could output the data in a formatted display that alerts the user to unexpected data patterns.

### 7.2 General Description

The following discussion uses an example of an SDA extension that invokes the MBX command to output a formatted display of the status of the mailbox devices in the system. The source file, MBX$SDA.C, is provided in SYS$EXAMPLES.

An SDA extension consists of a shareable image, in this case MBX$SDA.EXE, either located in the directory SYS$LIBRARY or found by translating the logical name MBX$SDA. It contains two universal symbols: SDA$EXTEND, the entry point; and SDA$EXTEND_VERSION, the address of a longword that contains the version of the interface used (in the format of major/minor ident), which allows SDA to confirm it has activated a compatible extension. The image contains at least two modules: MBX$SDA, the user-written module that defines the
two symbols and provides the code and data necessary to produce the desired formatted output; and SDA_EXTEND_VECTOR, which provides jackets for all of the callable SDA routines, and is found in SYSSLIBRARY:VMSS$VOLATILE_PRIVATE_INTERFACES.OLB. The user-written portion can be split into multiple modules.

Whenever SDA receives an unrecognized command, like "SDA> MBX", it attempts to activate the shareable image MBX$SDA at the SDA$EXTEND entry point. If you choose a command name that matches the abbreviation of an existing command, SDA can be forced to activate the extension using the "DO" command. For example, if you had an SDA extension called VAL$SDA, you could not activate it with a command like "SDA> VAL" as SDA would interpret that as an abbreviation of its VALIDATE command. But VAL$SDA can be activated by issuing "SDA> DO VAL".

With or without the "DO" prefix, the rest of the command line is passed to the extension; it is up to the extension to parse it. The example extension MBX$SDA includes support for commands of the form "SDA> MBX SUMMARY" and "SDA> MBX <address>" to demonstrate this. If the extension is invoked with no arguments, it should do no more than display a simple announcement message, or prompt for input. This assists in the debugging of the extension, as described in Section 7.4.

7.3 Detailed Description

This section describes how to compile, link, and invoke an SDA extension. It also describes the contents of an SDA extension.

7.3.1 Compiling and Linking an SDA Extension

The user-written module is only supported when written in Compaq C (minimum Version 5.2), following the pattern of the example extension, MBX$SDA.C. It should be compiled and linked using commands of the following form:

```
$ cc mbx$sda + alpha$library:sys$lib_c /library
$ link /share -
   mbx$sda.obj, -
   alpha$library:vms$volatile_private_interfaces /library, -
   sys$input /option
symbol_vector = (sda$extend=procedure)
symbol_vector = (sda$extend_version=data)
```

Note

1. You can include the qualifier /INSTRUCTION=NOFLOAT on the compile command line if floating-point instructions are not needed.
2. The + ALPHA$LIBRARY:SYSSLIB_C /LIBRARY is not needed on the compile command line if the logical name DECC$TEXT_LIBRARY is defined and translates to ALPHA$LIBRARY:SYSSLIB_C.TLB.
3. If the user-written extension needs to signal SDA condition codes, or output their text with $PUTMSG, you should add the qualifier /INCLUDE=SDAMSG to the parameter ALPHA$LIBRARY:VMS$VOLATILE_PRIVATE_INTERFACES /LIBRARY.
7.3.2 Invoking an SDA Extension

You can invoke the SDA extension as follows:

\$ define mbx$sda sys$disk::]mbx$sda
\$analyze /system
SDA>mbx summary
SDA>mbx <address>

7.3.3 Contents of an SDA Extension

At a minimum, the user-written module must contain:

- #include statements for DESCRIP.H and SDA_ROUTINES.H
- The global variable SDA$EXTEND_VERSION, initialized as follows:

\[
\text{int sda$extend\_version = SDA\_FLAGS$K\_VERSION;}
\]

- The routine SDA$EXTEND (prototype follows)

Optionally, the user-written module may also contain the statement:

\#define __NEW_STARLET

You should use this option because it provides type checking of function arguments and gives consistency in casing and naming conventions.

The entry point in the user-written module, SDA$EXTEND, is called as a routine with three arguments and no return value. The declaration is as follows:

\[
\text{void sda$extend (}
\text{\quad \quad \quad \quad \quad \quad \quad \quad int *transfer\_table,}
\text{\quad \quad \quad \quad \quad \quad \quad \quad struct dsc$descriptor\_s \*cmd\_line,}
\text{\quad \quad \quad \quad \quad \quad \quad \quad SDA\_FLAGS sda\_flags)}
\]

The arguments in this code example have the following meanings:
SDA Extension Routines
7.3 Detailed Description

<table>
<thead>
<tr>
<th>Line of Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>transfer_table</td>
<td>Address of the vector table in the base image. The user-written routine SDA$EXTEND must copy this to SDA$EXTEND_VECTOR_TABLE_ADDR before any SDA routines can be called.</td>
</tr>
<tr>
<td>cmd_line</td>
<td>Address of the descriptor of the command line as entered by the user, less the name of the extension. So, if you enter &quot;SDA&gt; MBX&quot; or &quot;SDA&gt; DO MBX&quot;, the command line is a zero length string. If you enter the command &quot;SDA&gt; MBX 80102030&quot;, the command line is &quot; 80102030&quot; (the separating space is not stripped).</td>
</tr>
<tr>
<td>sda_flags</td>
<td>Definition for the following four bits in this structure:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sda_flags.sda_flags$v_override</td>
<td>Indicates SDA has been activated with the ANALYZE/CRASH_DUMP/OVERRIDE command</td>
</tr>
<tr>
<td>sda_flags.sda_flags$v_current</td>
<td>Indicates SDA has been activated with the ANALYZE/SYSTEM command</td>
</tr>
<tr>
<td>sda_flags.sda_flags$v_target</td>
<td>Indicates that SDA was invoked from the kept debugger during an SCD or SDD session or when analyzing a process dump</td>
</tr>
<tr>
<td>sda_flags.sda_flags$v_process</td>
<td>Indicates SDA was activated with the ANALYZE/CRASH_DUMP command to analyze a process dump</td>
</tr>
</tbody>
</table>

The first executable statement of the routine must be to copy TRANSFER_TABLE to SDA$VECTOR_TABLE (which is declared in SDA_ROUTINES.H):

```
sda$vector_table = transfer_table;
```

If this is not done, you cannot call any of the routines described below. Any attempts to call the routines receive a status return of SDA$ VECNOTINIT. (For routines defined not to return a status, this value can be found only by examining R0.)

The next statement should be one to establish a condition handler, as it is often difficult to track down errors in extensions such as access violations because the extension is activated dynamically with LIB$FIND_IMAGE_SYMBOL. A default condition handler, SDA$COND_HANDLER, is provided that outputs the following information in the event of an error:

- The error condition
- The VMS version
- A list of activated images, with start and end virtual addresses
7.3 Detailed Description

- The signal array and register dump
- The current call frame chain

You can establish this condition handler as follows:

```c
lib$establish (sda$cond_handler);
```

--- Note

The error condition, signal array, and register dump are output directly to SYS$OUTPUT and/or SYS$ERROR, and are not affected by the use of the SDA commands SET OUTPUT and SET LOG.

---

Thus, a minimal extension would be:

```c
#define __NEW_STARLET 1
#include <descrip.h>
#include <sda_routines.h>
int sda$extend_version = SDA_FLAGS$K_VERSION;
void sda$extend (int *transfer_table,
                 struct dsc$descriptor_s *cmd_line,
                 SDA_FLAGS sda_flags)
{
    sda$vector_table = transfer_table;
    lib$establish (sda$cond_handler);
    sda$print ("hello, world");
    return;
}
```

7.4 Debugging an Extension

In addition to the "after-the-fact" information provided by the condition handler, you can debug SDA extensions using the OpenVMS Debugger. A second copy of the SDA image, SDA_DEBUG.EXE, is provided in SYS$SYSTEM. By defining the logical name SDA to reference this image, you can debug SDA extensions as follows:

- Compile your extension /DEBUG/NOOPT and link it /DEBUG.
- Define logical names for SDA and the extension, and invoke SDA.
- Type GO at the initial DBG> prompt.
- Invoke the extension with no argument at the initial SDA> prompt.
- Return control to Debug at the next prompt (either from SDA or the extension).
- Use Debug commands to set breakpoints, and so on, in the extension and then type GO.
- Invoke the extension, providing the necessary arguments.
An example of the previous procedures is as follows:

```bash
$ cc /debug /noopt mbx$sda + alpha$library:sys$lib_c /library
$ link /debug /share -
    mbx$sda.obj, -
    alpha$library:vms$volatile_private_interfaces /library, -
    sys$input /option
symbol_vector = (sda$extend=procedure)
symbol_vector = (sda$extend_version=data)
$ !
$ define mbx$sda sys$disk:].mbx$sda
$ define sda sda_debug
$ analyze /system
...
DBG> go
...
SDA> mbx
MBX commands: ’MBX SUMMARY’ and ’MBX <address>’
SDA>
”C <CR>
DBG> set image mbx$sda
DBG> set language c
DBG> set break /exception
DBG> go
SDA> mbx summary
...
SDA> mbx <address>
...
%DEBUG-I-DYNMODSET, setting module MBX$SDA
%SYSTEM-E-INVARG, invalid argument
...
DBG>
```

### 7.5 Callable Routines Overview

The user-written routine may call SDA routines to accomplish any of the following tasks:

- Read the contents of memory locations in the dump.
- Translate symbol names to values and vice-versa, define new symbols, and read symbol table files.
- Map an address to the activated image or executive image that contains that address.
- Output text to the terminal, with page breaks, page headings, and so on (and which is output to a file if the SDA commands SET OUTPUT or SET LOG have been used).
- Allocate and deallocate dynamic memory.
- Validate queues/lists.
- Format data structures.
- Issue any SDA command.
The full list of available routines is as follows:

SDA$ADD_SYMBOL  SDA$GETMEM
SDA$ALLOCATE    SDA$INSTRUCTION_DECODE
SDA$DBG_IMAGE_INFO SDA$NEW_PAGE
SDA$DEALLOCATE  SDA$PARSE_COMMAND
SDA$DISPLAY_HELP SDA$PRINT
SDA$ENSURE      SDA$READ_SYMFILE
SDA$FORMAT      SDA$REQMEM
SDA$FORMAT_HEADING SDA$SET_ADDRESS
SDA$GET_ADDRESS SDA$SET_CPU
SDA$GET_BLOCK_NAME SDA$SET_HEADING_ROUTINE
SDA$GET_BUGCHECK_MSG SDA$SET_LINE_COUNT
SDA$GET_CURRENT_CPU SDA$SET_PROCESS
SDA$GET_CURRENT_PCB SDA$SKIP_LINES
SDA$GET_HEADER  SDA$SYMBOL_VALUE
SDA$GET_HW_NAME SDA$SYMBOLIZE
SDA$GET_IMAGE_OFFSET SDA$TRYMEM
SDA$GET_INPUT SDA$TYPE
SDA$GET_LINE_COUNT SDA$VALIDATE_QUEUE

The details of all these routines follow. But there are some points to be aware of in using them:

• There are three different routines available to read the contents of memory locations in the dump: SDA$TRYMEM, SDA$GETMEM, and SDA$REQMEM. They are used as follows:

  SDA$TRYMEM is called from both SDA$GETMEM and SDA$REQMEM as the lower-level routine that actually does the work. SDA$TRYMEM returns success/failure status in R0, but does not signal any errors. Use it directly when you expect that the location being read is inaccessible. The caller of SDA$TRYMEM will handle this situation by checking the status returned by SDA$TRYMEM.

  SDA$GETMEM signals a warning when any error status is returned from SDA$TRYMEM. Signaling a warning will print out a warning message, but does not abort the SDA command in progress. You should use this routine when you expect the location to be read to be accessible. This routine does not prevent the command currently being executed from continuing. The caller of SDA$GETMEM must allow for this by checking the status returned by SDA$GETMEM.

  SDA$REQMEM signals an error when any error status is returned from SDA$TRYMEM. Signaling an error will print out an error message, abort the SDA command in progress and return to the “SDA>” prompt. You should use this routine when you expect the location to be read to be accessible. This routine will prevent the command currently being executed from continuing. The caller of SDA$REQMEM will not resume if an error occurs.

• You should use only the routines provided to output text. Do not use printf() or any other standard routine. If you do, the SDA commands SET OUTPUT and SET LOG will not produce the expected results. Do not include control characters in output (except tab); in particular, avoid <CR>,<LF>.
SDA Extension Routines
7.5 Callable Routines Overview

<LF>, <FF>, and the FAO directives that create them. Use the FAO directive !AF when contents of memory returned by SDA$TRYMEM, and so on, are being displayed directly, because embedded control characters will cause undesirable results. For example, displaying process names or resource names that contain particular control characters or escape sequences can lock up the terminal.

- You should use only the routines provided to allocate and deallocate dynamic memory. Do not use malloc() and free(). Where possible, allocate dynamic memory once, the first time the extension is activated, and deallocate it only if it needs to be replaced by a larger allocation. Because SDA commands can be interrupted by invoking another command at the "Press return for more" prompt, it is very easy to cause memory leaks.

- Some routines expect 32-bit pointers, and others expect 64-bit pointers. At first this not may appear to be logical, but in fact it is. All code and data used by SDA and any extensions must be in P0 or P1 space, as SDA does not need to (and does not) use P2 space for local data storage. However, addresses in the system dump (or running system, in the case of ANALYZE/SYSTEM) are 64-bit addresses, and SDA must provide access to all locations in the dump.

So, for example, the first two arguments to the routine SDA$TRYMEM are:

```
VOID_PQ start /* 64-bit pointer */
void *dest /* 32-bit pointer */
```

They specify the address of interest in the dump and the address in local storage to which the dump contents are to be copied.

7.6 Callable Routines Specifics

The following section describes the SDA extension callable routines.
SDA$ADD_SYMBOL

Adds a symbol to SDA's local symbol table.

Format

    void sda$add_symbol (char *symbol_name, uint64 symbol_value);

Arguments

    symbol_name
    OpenVMS usage char_string
type            character string
access          read only
mechanism       by reference

Address of symbol name string (zero-terminated).

    symbol_value
    OpenVMS usage quadword_unsigned
type             quadword (unsigned)
access           read only
mechanism        by value

The symbol value.

Description

SDA maintains a list of symbols and the corresponding values. SDA$ADD_SYMBOL is used to insert additional symbols into this list, so that they can be used in expressions and during symbolization.

Condition Values Returned

None

Example

    sda$add_symbol ("MBX", 0xFFFFFFFF80102030);

This call defines the symbol MBX to the hexadecimal value FFFFFFFF80102030.
SDA$ALLOCATE

Allocates dynamic memory.

Format

void sda$allocate (uint32 size, void **ptr_block);

Arguments

size
OpenVMS usage longword Unsigned
type longword (unsigned)
access read only
mechanism by value
Size of block to allocate (in bytes).

ptr_block
OpenVMS usage address
type longword (unsigned)
access write only
mechanism by reference
Address of longword to receive address of block.

Description

The requested memory is allocated and the address returned. Note that this is
the only supported mechanism for allocation of dynamic memory.

Related Routine
SDA$DEALLOCATE

Condition Values Returned

None
If no memory is available, the error is signaled and the SDA session aborted.

Example

PCB *local_pcb;
...
sda$allocate (PCB$C_LENGTH, (void *)&local_pcb);

This call allocates a block of heap storage for a copy of a PCB, and stores its
address in the pointer LOCAL_PCB.
SDA$DBG_IMAGE_INFO

Displays a list of activated images together with their virtual address ranges for debugging purposes.

Format

void sda$dbg_image_info ();

Arguments

None.

Description

A list of the images currently activated, with their start and end addresses, is displayed. This is provided as a debugging aid for SDA extensions.

Condition Values Returned

None

Example

sda$dbg_image_info ();

SDA outputs the list of images in the following format:

Current VMS Version: "X6DX-FT1"

Process Activated Images:

<table>
<thead>
<tr>
<th>Start VA</th>
<th>End VA</th>
<th>Image Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010000</td>
<td>000301FF</td>
<td>SDA</td>
</tr>
<tr>
<td>00032000</td>
<td>00177FFF</td>
<td>SDA$SHARE</td>
</tr>
<tr>
<td>07B508000</td>
<td>07B5BFFF</td>
<td>DECC$SHR</td>
</tr>
<tr>
<td>07B2D8000</td>
<td>07B399FFF</td>
<td>DEPC$SHR</td>
</tr>
<tr>
<td>07B288000</td>
<td>07B2C9FFF</td>
<td>CMA$TIS_SHR</td>
</tr>
<tr>
<td>07B698000</td>
<td>07B6D9FFF</td>
<td>LBR$RTL</td>
</tr>
<tr>
<td>00021A000</td>
<td>0025A3FF</td>
<td>SMG$SHR</td>
</tr>
<tr>
<td>000178000</td>
<td>002187FF</td>
<td>SMG$RTL</td>
</tr>
<tr>
<td>07B1E8000</td>
<td>07B239FFF</td>
<td>LIB$RTL</td>
</tr>
<tr>
<td>07B248000</td>
<td>07B279FFF</td>
<td>LIB$RTL</td>
</tr>
<tr>
<td>08C140D00</td>
<td>08C23120</td>
<td>SYSS$BASE_IMAGE</td>
</tr>
<tr>
<td>08C035888</td>
<td>08C052888</td>
<td>SYSS$PUBLIC_VECTORS</td>
</tr>
<tr>
<td>00026000</td>
<td>002D31FF</td>
<td>PRGDEV$MSG</td>
</tr>
<tr>
<td>002D4000</td>
<td>002DA9FF</td>
<td>SHR$MSG</td>
</tr>
<tr>
<td>002D0000</td>
<td>002DFF2FF</td>
<td>DECC$MSG</td>
</tr>
<tr>
<td>00380000</td>
<td>003E03FF</td>
<td>MBX$SDA</td>
</tr>
</tbody>
</table>
SDA Extension Routines
SDA$DEALLOCATE

SDA$DEALLOCATE

Deallocates and frees dynamic memory.

Format

void sda$deallocate (void *ptr_block, uint32 size);

Arguments

ptr_block
OpenVMS usage address
type longword (unsigned)
access read only
mechanism by value
Starting address of block to be freed.

size
OpenVMS usage longword unsigned
type longword (unsigned)
access read only
mechanism by value
Size of block to deallocate (in bytes).

Description

The specified memory is deallocated. Note that this is the only supported
mechanism for deallocation of dynamic memory.

Related Routine
SDA$ALLOCATE

Condition Values Returned

None
If an error occurs, it is signaled and the SDA session aborted.

Example

PCB *local_pcb;
...
sda$deallocate ((void *)local_pcb, PCB$C_LENGTH;
This call deallocates the block of length PCB$C_LENGTH whose address is stored
in the pointer LOCAL_PCB.
SDA$DISPLAY_HELP

Displays online help.

Format

void sda$display_help (char *library_desc, char *topic_desc);

Arguments

library
OpenVMS usage char_string
type character string
access read only
mechanism by reference
Address of library filespec. Specify as zero-terminated ASCII string.

topic
OpenVMS usage char_string
type character string
access read only
mechanism by reference
Address of topic name. Specify as zero-terminated ASCII string.

Description

Help from the specified library is displayed on the given topic.

Condition Values Returned

None

Example

sda$display_help ("SYS$HELP:SDA", "HELP");

This call produces the following output at the terminal:

HELP

The System Dump Analyzer (SDA) allows you to inspect the contents of memory as saved in the dump taken at crash time or as exists in a running system. You can use SDA interactively or in batch mode. You can send the output from SDA to a listing file. You can use SDA to perform the following operations:
Assign a value to a symbol
Examine memory of any process
Format instructions and blocks of data
Display device data structures
Display memory management data structures
Display a summary of all processes on the system
Display the SDA symbol table
Copy the system dump file
Send output to a file or device
Read global symbols from any object module
Send output to a file or device
Read global symbols from any object module
Search memory for a given value

For help on performing these functions, use the HELP command and specify a topic.

Format

HELP [topic-name]

Additional information available:
Parameter
HELP Subtopic?
SDA$ENSURE

Ensures sufficient space on the current output page.

Format

void sda$ensure (uint32 lines);

Argument

lines
OpenVMS usage longword unsigned
type longword (unsigned)
access read only
mechanism by value
Number of lines to fit on a page.

Description

This routine checks and makes sure that the number of lines specified fit on the current page; otherwise, it issues a page break.

Condition Values Returned

None

Example

sda$ensure (5);

This call ensures that there are five lines left on the current page, and it outputs a page break if there are not.
SDA Extension Routines
SDA$FORMAT

SDA$FORMAT

Displays the formatted contents of a data structure.

Format

void sda$format (VOID_PQ struct_addr, __optional_params);

Arguments

struct_addr
OpenVMS usage address
type quadword (unsigned)
access read only
mechanism by value

The address in the system dump of the data structure to be formatted.

options
OpenVMS usage mask_longword
type longword (unsigned)
access read only
mechanism by value

The following provides more information on options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Uses structure type from the xxx$B_TYPE field of the structure. This is the default.</td>
</tr>
<tr>
<td>SDA_OPT$M_FORMAT_TYPE</td>
<td>Uses the structure type given in struct_prefix.</td>
</tr>
<tr>
<td>SDA_OPT$M_FORMAT_PHYSICAL</td>
<td>Indicates that struct_addr is a physical address instead of a virtual address.</td>
</tr>
</tbody>
</table>

struct_prefix
OpenVMS usage char_string
type character string
access read only
mechanism by reference

Address of structure name string (zero-terminated).

Description

This routine displays the formatted content of a data structure that begins at the address specified. If no symbol prefix is passed, then SDA tries to find the symbols associated with the block type specified in the block-type byte of the data structure.
Condition Values Returned

None

Example

```c
PCB *local_pcb;
PHD *local_phd;
...
sda$format (local_pcb);
sda$format (local_phd, SDA_OPT$M_FORMAT_TYPE, "PHD");
```

The first call formats the structure whose system address is held in the variable LOCAL_PCB, determining the type from the type byte of the structure. The second call formats the structure whose system address is held in the variable LOCAL_PHD, using PHD symbols.
SDA$FORMAT_HEADING

Formats a new page heading.

Format

void sda$format_heading (char *ctrstr, __optional__params);

Arguments

ctrstr
OpenVMS usage char_string
type character-coded text string
access read only
mechanism by reference
Address of control string (zero-terminated ASCII string).

prmlst
OpenVMS usage varying_arg
type quadword (signed or unsigned)
access read only
mechanism by value
FAO parameters that are optional. All arguments after the control string are copied into a quadword parameter list as used by $FAOL_64.

Description

This routine prepares and saves the page heading to be used whenever SDA$NEW_PAGE is called. Nothing is output either until SDA$NEW_PAGE is next called, or a page break is necessary because the current page is full.

Condition Values Returned

None

If the $FAOL_64 call issued by SDA$FORMAT_HEADING fails, an empty string is used as the page heading.

Example

char hw_name[64];
...
sda$get_hw_name (hw_name, sizeof(hw_name));
sda$format_heading (  
    "SDA Extension Commands, system type !AZ",
    $hw_name);
sda$new_page ();

This example produces the following heading:

SDA Extension Commands, system type DEC 3000 Model 400
------------------------------------------------------
SDA$GET_ADDRESS

Gets the address value of the current memory location.

**Format**

```c
void sda$get_address (VOID_PQ *address);
```

**Argument**

- **address**
  - OpenVMS usage: quadword unsigned
  - type: quadword (unsigned)
  - access: write only
  - mechanism: by reference

Location to store the current 64-bit memory address.

**Description**

Returns the current address being referenced by SDA (location ".").

**Condition Values Returned**

None

**Example**

```c
VOID_PQ current_address;
...
sda$get_address (&current_address);
```

This call stores SDA's current memory location in the long pointer CURRENT_ADDRESS.
SDA Extension Routines
SDA$GET_BLOCK_NAME

SDA$GET_BLOCK_NAME

Returns the name of a structure, given its type and/or subtype.

Format

void sda$extend_get_block_name (uint32 block_type, uint32 block_subtype, char *buffer_ptr, uint32 buffer_len);

Arguments

block_type
OpenVMS usage longword unsigned
type longword (unsigned)
access read only
mechanism by value
Block type in range 0 - 255 (usually extracted from xxx$b_type field).

block_subtype
OpenVMS usage longword unsigned
type longword (unsigned)
access read only
mechanism by value
Block subtype in range 0 - 255 (ignored if the given block type has no subtypes).

buffer_ptr
OpenVMS usage char string
type character string
access write only
mechanism by reference
Address of buffer to save block name, which is returned as a zero-terminated string.

buffer_len
OpenVMS usage longword unsigned
type longword (unsigned)
access read only
mechanism by value
Length of buffer to receive block name.

Description

Given the block type and/or subtype of a structure, this routine returns the name of the structure. If the structure type is one that has no subtypes, the given subtype is ignored. If the structure type is one that has subtypes, and the subtype is given as zero, the name of the block type itself is returned. If an invalid type or subtype (out of range) is given, an empty string is returned.
Note

The buffer should be large enough to accommodate the largest possible block name (25 bytes plus the termination byte). The block name is truncated if it is too long for the supplied buffer.

Condition Values Returned

None

Example

```c
char buffer[32];
...
sda$get_block_name (0x6F, 0x20,
    buffer,
    sizeof (buffer));
if (strlen (buffer) == 0)
    sda$print ("Block type: no named type/subtype");
else
    sda$print ("Block type: !A2", buffer);

This example produces the following output:

Block type: VCC_CFCB
```
SDA$GET_BUGCHECK_MSG

Gets the text associated with a bugcheck code.

Format

void sda$get_bugcheck_msg (uint32 bugcheck_code, char *buffer_ptr, uint32 buffer_size);

Arguments

bugcheck_code
OpenVMS usage longword Unsigned
Type longword (unsigned)
Access read only
Mechanism by value
The bugcheck code to look up.

buffer_ptr
OpenVMS usage char_string
Type character string
Access write only
Mechanism by reference
Address of buffer to save bugcheck message.

buffer_len
OpenVMS usage longword Unsigned
Type longword (unsigned)
Access read only
Mechanism by value
Length of buffer to receive message.

Description

Gets the string representing the bugcheck code passed as the argument. The bugcheck message string is passed in the buffer (represented as a pointer and length) as a zero-terminated ASCII string.

Note

The buffer should be large enough to accommodate the largest possible bugcheck message (128 bytes including the termination byte). The text is terminated if it is too long for the supplied buffer.

Condition Values Returned

None
Example

```c
char buffer[128];
...
SDA$get_bugcheck_msg (0x108, buffer, sizeof(buffer));
SDA$print ("Bugcheck code 108 (hex) =");
SDA$print ("!_\"!AZ\"", buffer);

This example produces the following output:

Bugcheck code 108 (hex) =
  "DOUBLDALOC, Double deallocation of swap file space"
```
SDA$GET_CURRENT_CPU

 Gets the CPU database address of the currently selected CPU.

 Format

    void sda$get_current_cpu (cpu **cpudb);

 Arguments

    cpudb
    OpenVMS usage address
    type longword (unsigned)
    access write only
    mechanism by reference

    Location to which the address of the CPU database is to be returned.

 Description

    This routine causes SDA to return the address of the database for the currently
    selected CPU.

 Condition Values Returned

    None

 Example

    #include <cpudef>
    CPU *current_cpu;
    sda$get_current_cpu ( &current_cpu );

    In this example, the system address of the database for the current CPU is
    returned in variable current_cpu.
SDA$GET_CURRENT_PCB

Gets the PCB address of the “SDA current process” currently selected.

Format

```c
void sda$get_current_pcb (PCB **pcbadr);
```

Argument

- `pcbadr`
  - OpenVMS usage: quadword_unsigned
  - type: quadword (unsigned)
  - access: write only
  - mechanism: by reference

Location in which to store the current PCB address.

Description

The PCB address of the process currently selected by SDA is returned in the specified location.

Condition Values Returned

None

Example

```c
PCB *current_pcb;
...
}sda$get_current_pcb ( &current_pcb );
```

This call stores the system address of the PCB of the process currently being referenced by SDA in the pointer CURRENT_PCB.
SDA$GET_HEADER

Returns pointers to local copies of the dump file header and the error log buffer together with the sizes of those data structures.

Format

```c
void sda$get_header (DMP **dmp_header, uint32 *dmp_header_size, void **errlog_buf, uint32 *errlog_buf_size);
```

Arguments

**dmp_header**
OpenVMS usage: address  
type: longword (unsigned)  
access: write only  
mechanism: by reference  
Location in which to store the address of the copy of the dump file header held by SDA.

**dmp_header_size**
OpenVMS usage: longword_unsigned  
type: longword (unsigned)  
access: write only  
mechanism: by reference  
Location in which to store the size of the dump file header.

**errlog_buf**
OpenVMS usage: address  
type: longword (unsigned)  
access: write only  
mechanism: by reference  
Location in which to store the address of the copy of the error log buffer held by SDA.

**errlog_buf_size**
OpenVMS usage: longword_unsigned  
type: longword (unsigned)  
access: write only  
mechanism: by reference  
Location in which to store the size of the error log buffer.

Description

This routine returns the addresses and sizes of the dump header and error logs read by SDA when the dump file is opened. If this routine is called when the running system is being analyzed with ANALYZE/SYSTEM, then the following occurs:

- Returns the address and size of SDA's dump header buffer, but the header contains zeroes
- Returns zeroes for the address and size of SDA's error log buffer
Condition Values Returned

None

Example

DMP *dmp_header;
uint32 dmp_header_size;
char *errlog_buffer;
uint32 errlog_buffer_size;
...
sda$get_header (&dmp_header,
&dmp_header_size,
(void **) &errlog_buffer,
&errlog_buffer_size);

This call stores the address and size of SDA's copy of the dump file header in DMP_HEADER and DMP_HEADER_SIZE, and stores the address and size of SDA's copy of the error log buffers in ERRLOG_BUFFER and ERRLOG_BUFFER_SIZE, respectively.
SDA Extension Routines

SDA$GET_HW_NAME

Returns the full name of the hardware platform where the dump was written.

Format

void sda$get_hw_name (char *buffer_ptr, uint32 buffer_len);

Arguments

buffer_ptr
OpenVMS usage char_string
type character string
access write only
mechanism by reference
Address of buffer to save HW name.

buffer_len
OpenVMS usage longword_unsigned
type longword (unsigned)
access read only
mechanism by value
Length of buffer to receive HW name.

Description

Returns a zero-terminated ASCII string representing the platform hardware name and puts it in the buffer passed as the argument.

Note

The buffer should be large enough to accommodate the largest possible hardware platform name (120 bytes including the termination byte). The name is truncated if it is too long for the supplied buffer.

Condition Values Returned

None

Example

char hw_name[64];
...
sda$get_hw_name (hw_name, sizeof(hw_name));
sda$print ("Platform name: \"!AZ\", hw_name);

This example produces output of the form:

Platform name: "DEC 3000 Model 400"
SDA$GET_IMAGE_OFFSET

Maps a given virtual address onto an image or execlet.

Format

```
COMP_IMG_OFF sda$get_image_offset (VOID PQ va, VOID PQ img_info,
VOID PQ subimg_info, VOID PQ offset);
```

Arguments

**va**
- OpenVMS usage: address
- Type: quadword (unsigned)
- Access: read only
- Mechanism: by value
- Virtual address of interest.

**img_info**
- OpenVMS usage: address
- Type: quadword (unsigned)
- Access: write only
- Mechanism: by reference
- Pointer to return addr of LDRIMG or IMCB block.

**subimg_info**
- OpenVMS usage: address
- Type: quadword (unsigned)
- Access: write only
- Mechanism: by reference
- Pointer to return addr of ISD_OVERLAY or KFERES.

**offset**
- OpenVMS usage: quadword unsigned
- Type: quadword (unsigned)
- Access: write only
- Mechanism: by reference
- Pointer to address to return offset from image.

Description

Given a virtual address, this routine finds in which image it falls and returns the image information and offset. The loaded image list is traversed first to find this information. If it is not found, then the activated image list of the currently selected process is traversed. If still unsuccessful, then the resident installed images are checked.
SDA Extension Routines
SDA$GET_IMAGE_OFFSET

**Condition Values Returned**

<table>
<thead>
<tr>
<th>SDA_CIO$V_VALID</th>
<th>Set if image offset is found</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDA_CIO$V_PROCESS</td>
<td>Set if image is an activated image</td>
</tr>
<tr>
<td>SDA_CIO$V_SLICED</td>
<td>Set if the image is sliced</td>
</tr>
<tr>
<td>SDA_CIO$V_COMPRESSED</td>
<td>Set if activated image contains compressed data sections</td>
</tr>
<tr>
<td>SDA_CIO$V_ISD_INDEX</td>
<td>Index into ISD_LABELS table (only for LDRIMG execlets)</td>
</tr>
</tbody>
</table>

The status returned indicates the type of image if a match was found.

<table>
<thead>
<tr>
<th>SDA_CIO$V_xxx flags set:</th>
<th>img_info type:</th>
<th>subimg_info type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid</td>
<td>LDRIMG</td>
<td>n/a</td>
</tr>
<tr>
<td>valid &amp;&amp; sliced</td>
<td>LDRIMG</td>
<td>ISD_OVERLAY</td>
</tr>
<tr>
<td>valid &amp;&amp; process</td>
<td>IMCB</td>
<td>n/a</td>
</tr>
<tr>
<td>valid &amp;&amp; process &amp;&amp; sliced</td>
<td>IMCB</td>
<td>KFERES_SECTION</td>
</tr>
</tbody>
</table>

**Example**

```c
VOID_PQ va = (VOID_PQ)0xFFFFFFFF80102030;
COMP_IMG_OFF sda_cio;
int64 img_info;
int64 subimg_info;
int64 offset;
...
sda_cio = sda$get_image_offset (va,
    &img_info,
    &subimg_info,
    &offset);
```

For an example of code that interprets the returned COMP_IMG_OFF structure, see the supplied example program, SYS$EXAMPLES:MBX$SDA.C.
SDA$GET_INPUT

Reads input commands.

Format

```c
int sda$get_input (char *prompt, char *buffer, uint32 buflen);
```

Arguments

- **prompt**
  - OpenVMS usage: `char_string`
  - type: `character string`
  - access: `read only`
  - mechanism: `by reference`
  - Address of prompt string (zero-terminated ASCII string).

- **buffer**
  - OpenVMS usage: `char_string`
  - type: `character string`
  - access: `write only`
  - mechanism: `by reference`
  - Address of buffer to store command.

- **buflen**
  - OpenVMS usage: `longword unsigned`
  - type: `longword (unsigned)`
  - access: `read only`
  - mechanism: `by value`
  - Maximum length of buffer.

Description

The command entered is returned as a zero-terminated string. The string is not uppercased. If you do not enter input but simply press <return> or <ctrl/Z>, the routine returns a null string.

Condition Values Returned

- **SS$_NORMAL**  Successful completion.
- **RMS$_EOF**  User pressed <ctrl/Z>

Example

```c
int status;
char buffer[128];
...
status = sda$get_input ( "MBX> ", buffer, sizeof (buffer) );
```

This call prompts you for input with "MBX> " and stores the response in the buffer.
SDA Extension Routines
SDA$GET_LINE_COUNT

SDA$GET_LINE_COUNT

Obtains the number of lines currently printed on the current page.

Format

void sda$get_line_count (uint32 *line_count);

Argument

line_count
OpenVMS usage  longword unsigned
Type  longword (unsigned)
Access  write only
Mechanism  by reference

The number of lines printed on current page.

Description

Returns the number of lines that have been printed so far on the current page.

Condition Values Returned

None

Example

uint32 line_count;
...
sda$get_line_count (&line_count);

This call copies the current line count on the current page of output to the location LINE_COUNT.
SDA$GETMEM

Reads dump or system memory and signals a warning if inaccessible.

Format

```
int sda$getmem (VOID_PQ start, void *dest, int length, __optional__params);
```

Arguments

**start**
- OpenVMS usage: address
- type: quadword (unsigned)
- access: read only
- mechanism: by value

Starting virtual address in dump or system.

**dest**
- OpenVMS usage: address
- type: varies
- access: write only
- mechanism: by reference

Return buffer address.

**length**
- OpenVMS usage: longword unsigned
- type: longword (unsigned)
- access: read only
- mechanism: by value

Length of transfer.

**physical**
- OpenVMS usage: longword unsigned
- type: longword (unsigned)
- access: read only
- mechanism: by value

0: `<start>` is a virtual address. This is the default.
1: `<start>` is a physical address.

Description

This routine transfers an area from the memory in the dump file or the running system to the caller’s return buffer. It performs the necessary address translation to locate the data in the dump file. SDA$GETMEM signals a warning and returns an error status if the data is inaccessible.

Related Routines

SDA$REQMEM and SDA$TRYMEM
SDA Extension Routines
SDA$GETMEM

Condition Values Returned

<table>
<thead>
<tr>
<th>Condition Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDA$_SUCCESS</td>
<td>Successful completion</td>
</tr>
<tr>
<td>SDA$_NOREAD</td>
<td>The data is inaccessible for some reason.</td>
</tr>
<tr>
<td>SDA$_NOTINPHYS</td>
<td>The data is inaccessible for some reason.</td>
</tr>
<tr>
<td>Others</td>
<td>The data is inaccessible for some reason.</td>
</tr>
</tbody>
</table>

If a failure status code is returned, it has already been signaled as a warning.

Example

```c
int status;
PCB *current_pcb;
PHD *current_phd;
...
status = sda$getmem ((VOID_PQ)&current_pcb->pcb$l_phd, &current_phd, 4);
```

This call returns the contents of the PCB$L_PHD field of the PCB, whose system address is in the pointer CURRENT_PCB, to the pointer CURRENT_PHD.
SDA$INSTRUCTION_DECODE

Translates one Alpha machine instruction into the assembler string equivalent.

Format

```
int sda$instruction_decode (void *istream_ptr, char *buffer, uint32 buflen);
```

Arguments

**istream_ptr**  
OpenVMS usage address  
**type** longword (unsigned)  
**access** read/write  
**mechanism** by reference  
Address of the pointer that points to a copy of the i-stream in a local buffer.

**buffer**  
OpenVMS usage char_string  
**type** character string  
**access** write only  
**mechanism** by reference  
Address of a string buffer into which to store the output assembler string.

**buflen**  
OpenVMS usage longword unsigned  
**type** longword (unsigned)  
**access** read only  
**mechanism** by value  
Maximum size of the string buffer.

Description

Translates an Alpha machine instruction into the assembler string equivalent.

Alpha instructions are always 4 bytes long. The instruction stream must first be read into local memory and then the address of a pointer to the local copy of the instruction stream is passed to the routine. For every successful translated instruction, the pointer is automatically updated to point to the next instruction.

The output assembler string is zero-terminated and in case of a failure a null string is returned.

Condition Values Returned

```
SS$_NORMAL       Successful completion.
SS$_BADPARAM     Any of the following failures:
                  Output buffer too small
                  Invalid register
                  Invalid opcode class/format
                  Could not translate instruction
```

Status returns one of the above.
Example

```c
int status;
VOID_PQ va = (VOID_PQ)0xFFFFFFFF80102030;
uint32 instruction;
uint32 *istream = &instruction;
char buffer[64];
...
sda$reqmem (va, &instruction, 4);
status = sda$instruction_decode (&istream, buffer, sizeof (buffer));
```

This example reads the instruction at dump location VA and decodes it, putting the result into BUFFER. Pointer ISTREAM is incremented (to the next longword).
SDA$NEW_PAGE

Begins a new page of output.

Format

void sda$new_page ( );

Arguments

None.

Description

This routine causes a new page to be written and outputs the page heading (established with SDA$FORMAT_HEADING) and the current subheading (established with SDA$SET_HEADING_ROUTINE).

Condition Values Returned

None

Example

sda$new_page ();

This call outputs a page break and displays the current page heading and subheading (if any).
SDA Extension Routines
SDA$PARSE_COMMAND

SDA$PARSE_COMMAND

Parses and executes an SDA command line.

Format

void sda$parse_command (char *cmd_line, _ _optional_params);

Arguments

cmd_line
OpenVMS usage char_string
type character string
access read only
mechanism by reference
Address of a valid SDA command line (zero-terminated).

options
OpenVMS usage longword unsigned
type longword (unsigned)
access read only
mechanism by value

The options argument has the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDA_OPT$KPARSE_DONT_SAVE</td>
<td>Indicates &quot;do not save this command.&quot; This is the default.</td>
</tr>
<tr>
<td>SDA_OPT$KPARSE_SAVE</td>
<td>Indicates &quot;save this command.&quot; That is, it can be recalled with KP0 or REPEAT.</td>
</tr>
</tbody>
</table>

Description

Not every SDA command has a callable extension interface. For example, to redirect SDA's output, you would pass the command string "SET OUTPUT MBX.LIS" to this parse command routine. Abbreviations are allowed.

Condition Values Returned

None

Example

sda$parse_command ("SHOW ADDRESS 80102030");

This call produces the following output:
FFFFF00102030 is an S0/S1 address
  Mapped by Level-3 PTE at: FFFFFFFD.FFE00408
  Mapped by Level-2 PTE at: FFFFFFFD.FF7F800
  Mapped by Level-1 PTE at: FFFFFFFD.FF7FDFF8
  Mapped by Selfmap PTE at: FFFFFFFD.FF7FDFF0
  Also mapped in SPT window at: FFFFFFFF.FFDF0408

The "SHOW ADDRESS" command is not recorded as the most recent command for use with the KP0 key or the REPEAT command.
SDA$PRINT

Formats and prints a single line.

**Format**

```c
int sda$print (char *ctrstr, __optional_params);
```

**Arguments**

- **ctrstr**
  - OpenVMS usage: char_string
  - type: character-coded text string
  - access: read only
  - mechanism: by reference
  - Address of a zero-terminated control string.

- **prmlst**
  - OpenVMS usage: varying_arg
  - type: quadword (signed or unsigned)
  - access: read only
  - mechanism: by value
  - Optional FAO parameters. All arguments after the control string are copied into a quadword parameter list, as used by $FAOL_64.

**Description**

Formats and prints a single line. This is normally output to the terminal, unless you used the SDA commands SET OUTPUT or SET LOG to redirect or copy the output to a file.

**Condition Values Returned**

- SDA$_SUCCESS
  - Indicates a successful completion.
- SDA$_CNFLTARGS
  - Indicates more than twenty FAO parameters given.
- Other
  - Returns from the $PUT issued by SDA$PRINT (the error is also signaled). If the $FAOL_64 call issued by SDA$PRINT fails, a blank line is output.
Example

```c
char buffer[32];
...
sda$get_block_name (0x6F, 0x20,
    buffer,
    sizeof (buffer));
sda$print ("Block type: !AZ", buffer);

This example outputs the following line:

Block type: VCC_CFCB
```
SDA$READ_SYMFILE

Reads symbols from a given file.

Format

int sda$read_symfile (char *filespec, uint32 options, __optional_params);

Arguments

filespec
OpenVMS usage char_string
type character string
access read only
mechanism by reference

Address of file or directory specification from which to read the symbols (zero-terminated ASCII string).

options
OpenVMS usage longword_unsigned
type longword (unsigned)
access read only
mechanism by value

Indicates type of symbol file and flags, as shown in the following:

<table>
<thead>
<tr>
<th>Flags</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDA_OPT$M_READ_FORCE</td>
<td>read/force &lt;file&gt;</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_IMAGE</td>
<td>read/image &lt;file&gt;</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_SYMVA</td>
<td>read/symva &lt;file&gt;</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_RELO</td>
<td>read/relo &lt;file&gt;</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_EXEC</td>
<td>read/exec [&lt;dir&gt;]</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_NOLOG</td>
<td>/nolog, suppress count of symbols read</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_FILESPEC</td>
<td>&lt;file&gt; or &lt;dir&gt; given</td>
</tr>
<tr>
<td>SDA_OPT$M_READ_NOSIGNAL</td>
<td>return status, without signaling errors</td>
</tr>
</tbody>
</table>

relocate_base
OpenVMS usage address
type longword (unsigned)
access read only
mechanism by value

Base address for symbols (nonsliced symbols).

symvect_va
OpenVMS usage address
type longword (unsigned)
access read only
mechanism by value

The symbol vector address (symbols are offsets into the symbol vector).
**Description**

This command reads symbols from a given file to add symbol definitions to the working symbol table by reading GST entries. The file is usually a symbol file (.STB) or an image (.EXE). If SDA_OPT$M_READ_EXEC is specified in the options, then the filespec is treated as a directory specification, where symbol files and/or image files for all execlets may be found (à la READ/EXECUTIVE). If no directory specification is given, the logical name SDA$READ_DIR is used.

Note that when SDA reads symbol files and finds routine names, the symbol name that matches the routine name is set to the address of the procedure descriptor. A second symbol name, the routine name with “_C” appended, is set to the start of the routine's prologue.

**Condition Values Returned**

SDA$$_SUCCESS \quad$$ Successful completion.
SDA$$_CNFLTARGS \quad$$ No filename given and SDA_OPT$M_READ_EXEC not set.

Others errors are signaled and/or returned, exactly as though the equivalent SDA READ command had been used. Use HELP/MESSAGE for explanations.

**Example**

```
sda$read_symfile (*SDA$READ_DIR:SYSDEF*, SDA_OPT$M_READ_NOLOG);
```

The symbols in SYSDEF.STB are added to SDA's internal symbol table, and the number of symbols found is not output to the terminal.
SDA Extension Routines
SDA$REQMEM

**SDA$REQMEM**

Reads dump or system memory and signals an error if inaccessible.

**Format**

\[
\text{int sda$reqmem (VOID_PQ start, void *dest, int length, \_\_optional\_params);}\]

**Arguments**

**start**
- OpenVMS usage: address
- type: quadword (unsigned)
- access: read only
- mechanism: by value

Starting virtual address in dump or system.

**dest**
- OpenVMS usage: address
- type: varies
- access: write only
- mechanism: by reference

Return buffer address.

**length**
- OpenVMS usage: longword unsigned
- type: longword (unsigned)
- access: read only
- mechanism: by value

Length of transfer.

**physical**
- OpenVMS usage: longword unsigned
- type: longword (unsigned)
- access: read only
- mechanism: by value

0: <start> is a virtual address. This is the default.
1: <start> is a physical address.

**Description**

This routine transfers an area from the memory in the dump file or the running system to the caller's return buffer. It performs the necessary address translation to locate the data in the dump file. SDA$REQMEM signals an error and aborts the current command if the data is inaccessible.

**Related Routines**

SDA$GETMEM and SDA$TRYMEM
Condition Values Returned

SDA$_SUCCESS Successful completion.

Any failure is signaled as an error and the current command aborts.

Example

VOID_PQ address;
uint32 instruction;
...
sda$symbol_value ("EXE_STD$ALLOCATE_C", (uint64 *)&address);
sda$reqmem (address, &instruction, 4);

This example reads the first instruction of the routine EXE_STD$ALLOCATE into the location INSTRUCTION.
SDA Extension Routines
SDA$SET_ADDRESS

SDA$SET_ADDRESS

Stores a new address value as the current memory address (".").

Format

void sda$set_address (VOID_PQ address);

Argument

address
OpenVMS usage quadword_unsigned
type quadword (unsigned)
access read only
mechanism by value

Address value to store in current memory location.

Description

The specified address becomes SDA's current memory address (the predefined SDA symbol ".").

Condition Values Returned

None

Example

sda$set_address ((VOID_PQ)0xFFFFFFFF80102030);

This call sets SDA's current address to FFFFFFFF.80102030.
SDA$SET_CPU

Sets a new SDA CPU context.

Format

```
int sda$set_cpu (int cpu_id);
```

Arguments

- **cpu_id**
  - OpenVMS usage: longword_unsigned
  - type: longword (unsigned)
  - access: read only
  - mechanism: by value

  The desired CPU ID.

Description

This routine causes SDA to set the specified CPU as the currently selected CPU.

Condition Values Returned

- SDA$_SUCCESS: Successful completion.

  Any failure is signaled as an error and the current command aborts.

Example

```
int cpu_id = 2;
status = sda$set_cpu ( cpu_id );
```

In this example, SDA's current CPU context is set to the CPU whose number is held in the variable CPU_ID.
SDA Extension Routines
SDA$SET_HEADING_ROUTINE

SDA$SET_HEADING_ROUTINE

Sets the current heading routine to be called after each page break.

Format

void sda$set_heading_routine (void (*heading_rtn) ());

Argument

heading_rtn
OpenVMS usage procedure
type procedure value
access read only
mechanism by value

Address of routine to be called after each new page.

Description

When SDA begins a new page of output (either because SDA$NEW_PAGE was called, or because the current page is full), it outputs two types of headings. The first is the page title, and is set by calling the routine SDA$FORMAT_HEADING. This is the title that is included in the index page of a listing file when you issue a SET OUTPUT command. The second heading is typically for column headings, and as this can vary from display to display, you must write a routine for each separate heading. When you call SDA$SET_HEADING_ROUTINE to specify a user-written routine, the routine is called each time SDA begins a new page.

To stop the routine from being invoked each time SDA begins a new page, call either SDA$FORMAT_HEADING to set a new page title, or SDA$SET_HEADING_ROUTINE and specify the routine address as NULL.

If the column headings need to be output during a display (that is, in the middle of a page), and then be re-output each time SDA begins a new page, call the user-written routine directly the first time, then call SDA$SET_HEADING_ROUTINE to have it be called automatically thereafter.

Condition Values Returned

None
Example

```c
void mbx$title (void)
{
    sda$print ("Mailbox UCB ...");
    sda$print (" Unit Address ..." );
    sda$print ("------------------------");
    return;
}
...

sda$set_heading_routine (mbx$title);
...

sda$set_heading_routine (NULL);
```

This example sets the heading routine to the routine MBX$TITLE, and later clears it. The routine is called if any page breaks are generated by the intervening code.
**SDA$SET_LINE_COUNT**

Sets the number of lines printed so far on the current page.

**Format**

```c
void sda$set_line_count (uint32 line_count);
```

**Argument**

- **line_count**
  - OpenVMS usage: longword unsigned
  - type: longword (unsigned)
  - access: read only
  - mechanism: by value

  The number of lines printed on current page.

**Description**

The number of lines that have been printed so far on the current page is set to the given value.

**Condition Values Returned**

None

**Example**

```c
sda$set_line_count (5);
```

This call sets SDA's current line count on the current page of output to 5.
SDA$SET_PROCESS

Sets a new SDA process context.

Format

```c
int sda$set_process (const char *proc_name, int proc_index, int proc_addr);
```

Arguments

- **proc_name**
  - OpenVMS usage: character_string
  - type: character string
  - access: read only
  - mechanism: by reference
  - Address of the process name string (zero-terminated).

- **proc_index**
  - OpenVMS usage: longword_unsigned
  - type: longword (unsigned)
  - access: read only
  - mechanism: by value
  - The index of the desired process.

- **proc_addr**
  - OpenVMS usage: address
  - type: longword (unsigned)
  - access: read only
  - mechanism: by value
  - The address of the PCB for the desired process.

Description

This routine causes SDA to set the specified process as the currently selected process.

__________________________ Note ______________________________

The proc_name, proc_index, and proc_addr are mutually exclusive.

__________________________

Condition Values Returned

- **SDA$_SUCCESS**
  - Successful completion.

Any failure is signaled as an error and the current command aborts.

Example

```c
status = sda$set_process ( "JOB_CONTROL", 0, 0);
```

In this example, SDA's current process context is set to the JOB_CONTROL process.
SDA Extension Routines

SDA$SKIP_LINES

This routine outputs a specified number of blank lines.

Format

```c
void sda$skip_lines (uint32 lines);
```

Argument

```
lines
OpenVMS usage longword_unsigned
type longword (unsigned)
access read only
mechanism by value
```

Number of lines to skip.

Description

The specified number of blank lines are output.

Condition Values Returned

None

Example

```c
sda$skip_lines (2);
```

This call causes two blank lines to be output.
SDA$SYMBOL_VALUE

Obtains the 64-bit value of a specified symbol.

Format

```c
int sda$symbol_value (char *symb_name, uint64 *symb_value);
```

Arguments

**symb_name**
- OpenVMS usage: char_string
- type: character string
- access: read only
- mechanism: by reference
- Zero-terminated string containing symbol name.

**symb_value**
- OpenVMS usage: quadword_unsigned
- type: quadword (unsigned)
- access: write only
- mechanism: by reference
- Address to receive symbol value.

Description

A search through SDA’s symbol table is made for the specified symbol. If found, its 64-bit value is returned.

Condition Values Returned

- SDA$_SUCCESS Symbol found.
- SDA$_BADSYM Symbol not found.

Example

```c
int status;
VOID_PQ address;
...
status = sda$symbol_value ("EXE_STD$ALLOCATE_C", (uint64 *)&address);
This call returns the start address of the prologue of routine
EXE_STD$ALLOCATE to location ADDRESS.
```
SDA$SYMBOLIZE

Converts a value to a symbol name and offset.

Format

```c
int sda$symbolize (uint64 value, char *symbol_buf, uint32 symbol_len);
```

Arguments

- **value**
  - OpenVMS usage: quadword unsigned
  - Type: quadword (unsigned)
  - Access: read only
  - Mechanism: by value
  - Value to be translated.

- **symbol_buf**
  - OpenVMS usage: char_string
  - Type: character string
  - Access: write only
  - Mechanism: by reference
  - Address of buffer to which to return string.

- **symbol_len**
  - OpenVMS usage: longword unsigned
  - Type: longword (unsigned)
  - Access: read only
  - Mechanism: by value
  - Maximum length of string buffer.

Description

This routine accepts a value and returns a string that contains a symbol and offset corresponding to that value. First the value is checked in the symbol table. If no symbol can be found (either exact match or up to 0XFFF less than the specified value), the value is then checked to see if it falls within one of the loaded or activated images.

Condition Values Returned

- **SS$_NORMAL**
  - Successful completion.
- **SS$_BUFFEROVF**
  - Buffer too small, string truncated.
- **SS$_NOTRAN**
  - No symbolization for this value (null string returned).
Example

VOID_PQ va = VOID_PQ(0xFFFFFFFF80102030);
char buffer [64]
status = sda$symbolize (va, buffer, sizeof(buffer));
sda$print ("FFFFFFFF.80102030 = "!AZ"", buffer);

This example outputs the following:

FFFFFFFF.80102030 = "EXE$WRITE_PROCESS_C+00CD0"
SDA$TRYMEM

Reads dump or system memory and returns the error status (without signaling) if inaccessible.

Format

```c
int sda$trymem (VOID_PQ start, void *dest, int length, _ _optional_params);
```

Arguments

- `start`
  - OpenVMS usage: address
  - type: quadword (unsigned)
  - access: read only
  - mechanism: by value
  - Starting virtual address in dump or system.

- `dest`
  - OpenVMS usage: address
  - type: varies
  - access: write only
  - mechanism: by reference
  - Return buffer address.

- `length`
  - OpenVMS usage: longword_unsigned
  - type: longword (unsigned)
  - access: read only
  - mechanism: by value
  - Length of transfer.

- `physical`
  - OpenVMS usage: longword_unsigned
  - type: longword (unsigned)
  - access: read only
  - mechanism: by value
  - 0: `<start>` is a virtual address. This is the default.
  - 1: `<start>` is a physical address.

Description

This routine transfers an area from the memory in the dump file or the running system to the caller’s return buffer. It performs the necessary address translation to locate the data in the dump file. SDA$TRYMEM does not signal any warning or errors. It returns the error status if the data is inaccessible.

Related Routines

SDA$GETMEM and SDA$REQMEM
Condition Values Returned

SDA$_SUCCESS  Successful completion.
SDA$_NOREAD  The data is inaccessible for some reason.
SDA$_NOTINPHYS The data is inaccessible for some reason.
Others  The data is inaccessible for some reason.

Example

int status;
DDB *ddb;
...
status = sda$trymem (ddb->ddb$ps_link, ddb, DDB$K_LENGTH);
if ($VMS_STATUS_SUCCESS (status))
   sda$print ("Next DDB is successfully read from dump");
else
   sda$print ("Next DDB is inaccessible");

This example attempts to read the next DDB in the DDB list from the dump.
SDA$TYPE

Formats and types a single line to SYS$OUTPUT.

Format

int sda$type (char *ctrstr, _optional_params);

Arguments

ctrstr
OpenVMS usage char_string
type character-coded text string
access read only
mechanism by reference

Address of a zero-terminated control string.

prmlst
OpenVMS usage varying_arg
type quadword (signed or unsigned)
access read only
mechanism by value

Optional FAO parameters. All arguments after the control string are copied into
a quadword parameter list, as used by $FAOL_64.

Description

Formats and prints a single line to the terminal. This is unaffected by the use of
the SDA commands SET OUTPUT or SET LOG.

Condition Values Returned

SDA$_SUCCESS Indicates a successful completion.
SDA$_CNFLTARGS Indicates more than twenty FAO parameters
given.

Other Returns from the $PUT issued by SDA$TYPE
(the error is also signaled). If the $FAOL_64
call issued by SDA$TYPE fails, a blank line is
output.

Example

int status;
...
status = sda$type ("Invoking SHOW SUMMARY to output file...");

This example displays the message "Invoking SHOW SUMMARY to output file..."
to the terminal.
SDA$VALIDATE_QUEUE

Validates queue structures.

Format

void sda$validate_queue (VOID_PQ queue_header, __optional_params);

Arguments

queue_header
OpenVMS usage address
type quadword (unsigned)
access read only
mechanism by value
Address from which to start search.

options
OpenVMS usage mask_longword
type longword (unsigned)
access read only
mechanism by value

The following table shows the flags that indicate the type of queue:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Defaults to doubly-linked longword queue</td>
</tr>
<tr>
<td>SDA_OPT$M_QUEUE_BACKLINK</td>
<td>Validates the integrity of a doubly-linked queue using the back links instead of the forward links</td>
</tr>
<tr>
<td>SDA_OPT$M_QUEUE_LISTQUEUE</td>
<td>Displays queue elements for debugging</td>
</tr>
<tr>
<td>SDA_OPT$M_QUEUE_QUADLINK</td>
<td>Indicates a quadword queue</td>
</tr>
<tr>
<td>SDA_OPT$M_QUEUE_SELF</td>
<td>Indicates a self-relative queue</td>
</tr>
<tr>
<td>SDA_OPT$M_QUEUE_SINGLINK</td>
<td>Indicates a singly-linked queue</td>
</tr>
</tbody>
</table>

Description

You can use this routine to validate the integrity of double-linked, single-linked or self-relative queues either with longword or quadword links. If you specify the option SDA_OPT$M_QUEUE_LISTQUEUE, the queue elements are displayed for debugging. Otherwise a one-line summary indicates how many elements were found and whether the queue is intact.

Condition Values Returned

None

If an error occurs, it is signaled by SDA$VALIDATE_QUEUE.
Example

```c
int64 temp;
int64 *queue;
...
sda$symbol_value ("EXE$GL_NONPAGED", &temp);
temp += 4;
sda$reqmem ((VOID_PQ)temp, &queue, 4);
sda$validate_queue (queue, SDA_OPT$M_QUEUE_SINGLINK);
```

This sequence validates the nonpaged pool free list, and outputs a message of the form:

```
Queue is zero-terminated, total of 204 elements in the queue
```
Part II

OpenVMS Alpha System Code Debugger & System Dump Debugger

Part II describes the System Code Debugger (SCD) and the System Dump Debugger (SDD). It presents how to use SCD and SDD by doing the following:

• Building a system image to be debugged
• Setting up the target system for connections
• Setting up the host system
• Starting SCD
• Troubleshooting connections and network failures
• Looking at a sample SCD session
• Analyzing memory as recorded in a system dump
• Looking at a sample SDD session
This chapter describes the OpenVMS Alpha System Code Debugger (SCD) and how it can be used to debug non pageable system code and device drivers running at any interrupt priority level (IPL).

You can use SCD to perform the following tasks:

- Control the system software’s execution—stop at points of interest, resume execution, intercept fatal exceptions, and so on
- Trace the execution path of the system software
- Monitor exception conditions
- Examine and modify the values of variables
- Test the effect of modifications, in some cases, without having to edit the source code, recompile, and relink

The use of SCD requires two systems:

- The host system, probably also the system where the image to be debugged has been built
- The target system, usually a standalone test system, where the image being debugged is executed

SCD is a symbolic debugger. You can specify variable names, routine names, and so on, precisely as they appear in your source code. SCD can also display the source code where the software is executing, and allow you to step by source line.

SCD recognizes the syntax, data typing, operators, expressions, scoping rules, and other constructs of a given language. If your code or driver is written in more than one language, you can change the debugging context from one language to another during a debugging session.

To use SCD, you must do the following:

- Build a system image or device driver to be debugged.
- Set up the target kernel on a standalone system.
  - The target kernel is the part of SCD that resides on the system that is being debugged. It is integrated with XDELTA and is part of the SYSTEM_DEBUG execlet.
- Set up the host system environment, which is integrated with the OpenVMS Debugger.

The following sections cover these tasks in more detail, describe the available user-interface options, summarize applicable OpenVMS Debugger commands, and provide a sample SCD session.
8.1 User-Interface Options

SCD has the following user-interface options:

- A DECwindows Motif interface for workstations
  When using this interface, you interact with SCD by using a mouse and pointer to choose items from menus, click on buttons, select names in windows, and so on.
  Note that you can also use OpenVMS Debugger commands with the DECwindows Motif interface.

- A character cell interface for terminals and workstations
  When using this interface, you interact with SCD by entering commands at a prompt. The sections in this chapter describe how to use the system code debugger with the character cell interface.

For more information about using the OpenVMS DECwindows Motif interface and OpenVMS Debugger commands with SCD, see the OpenVMS Debugger Manual.

8.2 Building a System Image to Be Debugged

1. Compile the sources you want to debug, and be sure to use the /DEBUG and /NOOPT qualifiers.

   Note
   Debugging optimized code is much more difficult and is not recommended unless you know the Alpha architecture well. The instructions are reordered so much that single-stepping by source line will look like you are randomly jumping all over the code. Also note that you cannot access all variables. SCD reports that they are optimized away.

2. Link your image using the /DSF (debug symbol file) qualifier. Do not use the /DEBUG qualifier, which is for debugging user programs. The /DSF qualifier takes an optional filename argument similar to the /EXE qualifier. For more information, see the OpenVMS Linker Utility Manual. If you specify a name in the /EXE qualifier, you will need to specify the same name for the /DSF qualifier. For example, you would use the following command:

   $ LINK/EXE=EXE$:MY_EXECLET/DSF=EXE$:MY_EXECLET OPTIONS_FILE/OPT

   The .DSF and .EXE file names must be the same. Only the extensions will be different, that is .DSF and .EXE.
   The contents of the .EXE file should be exactly the same as if you had linked without the /DSF qualifier. The .DSF file will contain the image header and all the debug symbol tables for .EXE file. It is not an executable file, and cannot be run or loaded.

3. Put the .EXE file on your target system.

4. Put the .DSF file on your host system, because when you use SCD to debug code in your image, it will try to look for a .DSF file first and then look for an .EXE file. The .DSF file is better because it has symbols in it. Section 8.4 describes how to tell SCD where to find your .DSF and .EXE files.
8.3 Setting Up the Target System for Connections

The target kernel is controlled by flags and devices specified when the system is booted, by XDELTA commands, by a configuration file, and by several system parameters. The following sections contain more information about these items.

Boot Command

The form of the boot command varies depending on the type of OpenVMS Alpha system you are using. However, all boot commands have the concept of boot flags and boot devices as well as a way to save the default boot flags and devices. This section uses syntax from a DEC 3000 Model 400 Alpha Workstation in examples.

To use SCD, you must specify an Ethernet device with the boot command on the target system. The target system uses this device to communicate with the host debugger. It is currently a restriction that this device must not be used for anything else (either for booting or network software such as DECnet, TCP/IP products, and LAT products). Thus, you must also specify a different device from which to boot. For example, the following command will boot a DEC 3000 Model 400 from the DKB100 disk, and SCD will use the ESA0 Ethernet device.

```plaintext
>>> boot dkb100,esa0
```

To find out the Ethernet devices available on your system, enter the following command:

```plaintext
>>> show device
```

In addition to devices, you can also specify flags on the boot command line. Boot flags are specified as a hex number; each bit of the number represents a true or false value for a flag. The following flag values are relevant to the system code debugger.

- **8000**
  This is the SCD boot flag. It enables operation of the target kernel. If this SCD boot flag is not set, not only will it be impossible to use SCD to debug the system, but the additional XDELTA commands related to the target kernel will generate an XDELTA error message. If this boot flag is set, SYSTEM_DEBUG is loaded, and SCD is enabled.

- **0004**
  This is the initial breakpoint boot flag. It controls whether the system calls INI$BRK at the beginning and end of EXEC_INIT. Notice that if SCD is the default debugger, the first breakpoint is not as early as it is for XDELTA. It is delayed until immediately after the PFN database is set up.

- **0002**
  This is the XDELTA boot flag, which controls whether XDELTA is loaded. It behaves slightly differently when the SCD boot flag is also set.

If the SCD boot flag is clear, this flag simply determines if XDELTA is loaded. If the SCD boot flag is set, this flag determines whether XDELTA or the system code debugger is the default debugger. If the XDELTA flag is set, XDELTA will be the default debugger. In this state, the initial system breakpoints and any calls to INI$BRK trigger XDELTA, and you must enter an XDELTA command to start using SCD. If the XDELTA boot flag is clear, the initial breakpoints and calls to INI$BRK go to SCD. You cannot use XDELTA if the XDELTA boot flag is clear.
8.3 Setting Up the Target System for Connections

**Boot Command Example** The following command boots a DEC 3000 Model 400 from disk DKA0, enables SCD, defaults to using XDELTA, and takes the initial system boot breakpoints.

```bash
>>> boot dka0,esa0 -fl 0,8006
```

You can set these devices and flags to be the default values so that you will not have to specify them each time you boot the system. On a DEC 3000 Model 400, use the following commands:

```bash
>>> set bootdef_dev dka0,esa0
>>> set boot_osflags 0,8006
```

**SCD Configuration File**
The SCD target system reads a configuration file in SYS$SYSTEM named DBGTK$CONFIG.SYS. The first line of this file contains a default password, which must be specified by the host debug system to connect to the target. The default password may be the null string; in this case the host must supply the null string as the password (/PASSWORD="") on the connect command as described in Section 8.5, or no password at all. Other lines in this file are reserved by Compaq. Note that you must create this file because Compaq does not supply it. If this file does not exist, you can only run SCD by specifying a default password with the XDELTA ;R command described in the following section.

**XDELTA Commands**
When the system is booted with both the XDELTA boot flag and the SCD boot flag, the following two additional XDELTA commands are enabled:

- n,xxxx;R ContRol SCD connection
  
  You can use this command to do the following:
  - Change the password which the SCD host must present
  - Disconnect the current session from SCD
  - Give control to SCD by simulating a call to INI$BRK
  - Any combination of these

Optional string argument xxxx specifies the password that the system code debugger must present for its connection to be accepted. If this argument is left out, the required password is unchanged. The initial password is taken from the first line of the SYSSYSTEM:DBGTK$CONFIG.SYS file. The new password does not remain in effect across a boot of the target system.
The optional integer argument \( n \) controls the behavior of the ;R command as follows:

<table>
<thead>
<tr>
<th>Value of ( n )</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>Gives control to SCD by simulating a call to INI$BRK</td>
</tr>
<tr>
<td>+2</td>
<td>Returns to XDELTA after changing the password. 2;R without a password is a no-op</td>
</tr>
<tr>
<td>0</td>
<td>Performs the default action</td>
</tr>
<tr>
<td>-1</td>
<td>Changes the password, breaks any existing connection to SCD, and then simulates a call to INI$BRK (which will wait for a new connection to be established and then give control to SCD)</td>
</tr>
<tr>
<td>-2</td>
<td>Returns to XDELTA after changing the password and breaking an existing connection</td>
</tr>
</tbody>
</table>

Currently, the default action is the same action as +1.

If SCD is already connected, the ;R command transfers control to SCD, and optionally changes the password that must be presented the next time a system code debugger tries to make a connection. This new password does not last across a boot of the target system.

- \( n;K \) Change inibrK behavior
  
  If optional argument \( n \) is 1, future calls to INI$BRK will result in a breakpoint being taken by SCD. If the argument is 0, or no argument is specified, future calls to INI$BRK will result in a breakpoint being taken by XDELTA.

**SYSTEM Parameters**

- **DBGTK_SCRATCH**
  
  Bits 0 through 7 specify how many pages of memory are allocated for SCD. This memory is allocated only if system code debugging is enabled with the SCD boot flag (described earlier in this section). Usually, the default value of 1 is adequate; however, if SCD displays an error message, increase this value.
  
  Bits 8 through 31 are reserved by Compaq.

- **SCSNODE**
  
  Identifies the target kernel node name for SCD. See Section 8.3.1 for more information.

### 8.3.1 Making Connections Between the Target Kernel and the System Code Debugger

It is always SCD on the host system that initiates a connection to the target kernel. When SCD initiates this connection, the target kernel accepts or rejects the connection based on whether the remote debugger presents it with a node name and password that matches the password in the target system (either the default password from the SYS$SYSTEM:DBGTK$CONFIG.SYS file, or a different password specified via XDELTA). SCD obtains the node name from the SCSNODE system parameter.

The target kernel can accept a connection from SCD any time the system is running below IPL 22, or if XDELTA is in control (at IPL 31). However, the target kernel actually waits at IPL 31 for a connection from the SCD host in two cases: when it has no existing connection to an SCD host and (1) it
receives a breakpoint caused by a call to INI$BRK (including either of the initial breakpoints), or (2) when you enter a 1;R or -1;R command to XDELTA.

8.3.2 Interactions Between XDELTA and the Target Kernel/System Code Debugger

XDELTA and the target kernel are integrated into the same system. Normally, you choose to use one or the other. However, XDELTA and the target kernel can be used together. This section explains how they interoperate.

The XDELTA boot flag controls which debugger (XDELTA or the SCD target kernel) gets control first. If it is not set, the target kernel gets control first, and it is not possible to use XDELTA without rebooting. If it is set, XDELTA gets control first, but you can use XDELTA commands to switch to the target kernel and to switch INI$BRK behavior such that the target kernel gets control when INI$BRK is called.

Breakpoints always stick to the debugger that set them; for example, if you set a breakpoint at location “A” with XDELTA, and then you enter the commands 1;K (switch INI$BRK to the system code debugger) and ;R (start using the system code debugger) then, from SCD, you can set a breakpoint at location “B”. If the system executes the breakpoint at A, XDELTA reports a breakpoint, and SCD will see nothing (though you could switch to SCD by issuing the XDELTA ;R command). If the system executes the breakpoint at B, SCD will get control and report a breakpoint (you cannot switch to XDELTA from SCD).

Notice that if you examine location A with SCD, or location B with XDELTA, you will see a BPT instruction, not the instruction that was originally there. This is because neither debugger has any information about the breakpoints set by the other debugger.

One useful way to use both debuggers together is when you have a system that exhibits a failure only after hours or days of heavy use. In this case, you can boot the system with SCD enabled (8000), but with XDELTA the default (0002) and with initial breakpoints enabled (0004). When you reach the initial breakpoint, set an XDELTA breakpoint at a location that will only be reached when the error occurs. Then proceed. When the error breakpoint is reached, possibly days later, then you can set up a remote system to debug it and enter the ;R command to XDELTA to switch control to SCD.

Here is another technique to use when you do not know where to put an error breakpoint as previously mentioned. Boot the system with only the SCD boot flag set. When you see that the error has occurred, halt the system and initiate an IPL 14 interrupt, as you would to start XDELTA. The target kernel will get control and wait for a connection for SCD.

8.4 Setting Up the Host System

To set up the host system, you need access to all system images and drivers that are loaded (or can be loaded) on the target system. You should have access to a source listings kit or a copy of the following directories:

SYS$LOADABLE_IMAGES:
SYS$LIBRARY:
SYS$MESSAGE:

You need all the .EXE files in those directories. The .DSF files are available with the OpenVMS Alpha source listings kit.
Optionally, you need access to the source files for the images to be debugged. SCD will look for the source files in the directory where they were compiled. If your build system and host system are different, you must use the SET SOURCE command to point SCD to the location of the source code files. For an example of the SET SOURCE command, see Section 8.12.

Before making a connection to the target system, you must set up the logical name DBGHK$IMAGE_PATH, which must be set up as a search list to the area where the system images or .DSF files are kept. For example, if the copies are in the following directories:

DEVICE:[SYS$LDR]
DEVICE:[SYSLIB]
DEVICE:[SYSMSG]

you would define DBGHK$IMAGE_PATH as follows:

$ define dbghk$image_path DEVICE:[SYS$LDR],DEVICE:[SYSLIB],DEVICE:[SYSMSG]

This works well for debugging using all the images normally loaded on a given system. However, you might be using the debugger to test new code in an execlet or a new driver. Because that image is most likely in your default directory, you must define the logical name as follows:

$ define dbghk$image_path [],DEVICE:[SYS$LDR],DEVICE:[SYSLIB],DEVICE:[SYSMSG]

If SCD cannot find one of the images through this search path, a warning message is displayed. SCD will continue initialization as long as it finds at least one image. If SCD cannot find the SYS$BASE_IMAGE file, which is the OpenVMS Alpha operating system’s main image file, an error message is displayed and the debugger exits.

If and when this happens, check the directory for the image files and compare it to what is loaded on the target system.

### 8.5 Starting the System Code Debugger

To start SCD on the host side, enter the following command:

$ DEBUG/KEEP

SCD displays the DBG> prompt. With the DBGHK$IMAGE_PATH logical name defined, you can invoke the CONNECT command and the optional qualifiers /PASSWORD and /IMAGE_PATH.

To use the CONNECT command and the optional qualifiers (/PASSWORD and /IMAGE_PATH) to connect to the node with name <node-name>, enter the following command:

DBG> CONNECT %NODE_NAME node-name /PASSWORD="password"

If a password has been set up on the target system, you must use the /PASSWORD qualifier. If a password is not specified, a zero length string is passed to the target system as the password.

The /IMAGE_PATH qualifier is also optional. If you do not use this qualifier, SCD uses the DBGHK$IMAGE_PATH logical name as the default. The /IMAGE_PATH qualifier is a quick way to change the logical name. However, when you use it, you cannot specify a search list. You can use only a logical name or a device and directory, although the logical name can be a search list.
8.5 Starting the System Code Debugger

Usually, SCD obtains the source file name from the object file. This is put there by the compiler when the source is compiled with the /DEBUG qualifier. The SET SOURCE command can take a list of paths as a parameter. It treats them as a search list.

8.6 Summary of System Code Debugger Commands

In general, any OpenVMS debugger command can be used in SCD. For a complete list, refer to the OpenVMS Debugger Manual. The following are a few examples:

- Commands to manipulate the source display, such as TYPE and SCROLL.
- Commands used in OpenVMS debugger command programs, such as DO and IF.
- Commands that affect output formats, such as SET RADIX.
- Commands that manipulate symbols and scope, such as EVALUATE, SET LANGUAGE, and CANCEL SCOPE. Note that the debugger SHOW IMAGE command is equivalent to the XDELTA ;L command, and the debugger DEFINE command is equivalent to the XDELTA ;X command.
- Commands that cause code to be executed, such as STEP and GO. Note that the debugger STEP command is equivalent to the XDELTA S and O commands, and the debugger GO command is equivalent to the XDELTA ;P and ;G commands.
- Commands that manipulate breakpoints, such as SET BREAK and CANCEL BREAK. These commands are equivalent to the XDELTA ;B command. However, unlike XDELTA, there is no limit on the number of breakpoints in SCD.
- Commands that affect memory, such as DEPOSIT and EXAMINE. These commands are equivalent to the XDELTA /,!,[,]' commands.

You can also use the OpenVMS debugger command SDA to examine the target system with System Dump Analyzer semantics. This command, which is not available when debugging user programs, is described in the next section.

8.7 Using System Dump Analyzer Commands

Once a connection has been established to the target system, you can use the commands listed in the previous section to examine the target system. You can also use some System Dump Analyzer (SDA) commands, such as SHOW SUMMARY and SHOW DEVICE. This feature allows the system programmer to take advantage of the strengths of both the OpenVMS Debugger and SDA to examine the state of the target system and to debug system programs such as device drivers.

To obtain access to SDA commands, you simply type "SDA" at the OpenVMS Debugger prompt ("DBG>") at any time after a connection has been established to the target system. SDA initializes itself and then outputs the "SDA>" prompt. Enter SDA commands as required. (See Chapter 4 for more information.) To return to the OpenVMS Debugger, you enter "EXIT" at the "SDA>" prompt. Optionally, you may invoke SDA to perform a single command and then return immediately to the OpenVMS Debugger, as in the following example:

```
DBG> SDA SHOW SUMMARY
```
You may reenter SDA at any time, with or without the optional SDA command. Once SDA has been initialized, the SDA> prompt is output more quickly on subsequent occasions.

Note that there are some limitations on the use of SDA from within SCD.

- You cannot switch between processes, whether requested explicitly (SET PROCESS <name>) or implicitly (SHOW PROCESS <name>). The exception to this is that access to the system process is possible.

- You cannot switch between CPUs.

- SDA has no knowledge of the OpenVMS debugger’s Motif or Windows interfaces. Therefore, all SDA input and output occurs at the terminal or window where the OpenVMS debugger was originally invoked. Also, while using SDA, the OpenVMS debugger window is not refreshed; you must exit SDA to allow the OpenVMS debugger window to be refreshed.

- When you invoke SDA from SCD with an immediate command, and that command produces a full screen of output, SDA displays the message "Press RETURN for more." followed by the "SDA>" prompt before continuing. If you enter another SDA command at this prompt, SDA does not automatically return to SCD upon completion. To do this, you must enter an EXIT command.

8.8 System Code Debugger Network Information

The SCD host and the target kernel use a private Ethernet protocol to communicate. For the two systems to see each other, they have to be on the same Ethernet segment.

The network portion of the target system finds the first Ethernet device and communicates through it. The network portion of the host system also finds the first Ethernet device and communicates through it. However, if for some reason, SCD picks the wrong device, you can override this by defining the logical DBGHK$ADAPTOR to the template device name for the appropriate adaptor.

8.9 Troubleshooting Checklist

If you have trouble starting a connection, perform the following tasks to correct the problem:

- Check SCSNODE on the target system.
  It must match the name you are using in the host CONNECT command.

- Make sure that both the Ethernet and boot device are on the boot command.

- Make sure that the host system is using the correct Ethernet device, and that the host and target systems are connected to the same Ethernet segment.

- Check the version of the operating system and make sure that both the host and target systems are running the same version of the OpenVMS Alpha operating system.
8.10 Troubleshooting Network Failures

There are three possible network errors:

- **NETRETRY**
  Indicates the system code debugger connection is lost

- **SENDRETRY**
  Indicates a message send failure

- **NETFAIL**
  Results from the two previous errors

The netfail error message has a status code that can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 4, 6</td>
<td>Internal network error, submit a problem report to Compaq.</td>
</tr>
<tr>
<td>8, 10, 14, 16, 18, 20, 26, 28, 34, 38</td>
<td>Network protocol error, submit a problem report to Compaq.</td>
</tr>
<tr>
<td>22, 24</td>
<td>Too many errors on the network device most likely due to congestion. Reduce the network traffic or switch to another network backbone.</td>
</tr>
<tr>
<td>30</td>
<td>Target system scratch memory not available. Check DBGTK_SCRATCH. If increasing this value does not help, submit a problem report to Compaq.</td>
</tr>
<tr>
<td>32</td>
<td>Ran out of target system scratch memory. Increase value of DBGTK_SCRATCH.</td>
</tr>
<tr>
<td>All others</td>
<td>There should not be any other network error codes printed. If one occurs that does not match the previous ones, submit a problem report to Compaq.</td>
</tr>
</tbody>
</table>

8.11 Access to Symbols in OpenVMS Executive Images

Accessing OpenVMS executive images’ symbols is not always straightforward with SCD. Only a subset of the symbols may be accessible at one time and in some cases, the symbol value the debugger currently has may be stale. To understand these problems and their solutions, you must understand how the debugger maintains its symbol tables and what symbols exist in the OpenVMS executive images. The following sections briefly summarize these topics.

8.11.1 Overview of How the OpenVMS Debugger Maintains Symbols

The debugger can access symbols from any image in the OpenVMS loaded system image list by reading in either the .DSF or .EXE file for that particular image. The .EXE file contains information only about symbols that are part of the symbol vector for that image. The current image symbols for any set module are defined. (You can tell if you have the .DSF or .EXE file by doing a SHOW MODULE. If there are no modules, you have the .EXE file.) This includes any symbols in the SYS$BASE_IMAGE.EXE symbol vector for which the code or data resides in the current image. However, you cannot access a symbol that is part of the SYS$BASE_IMAGE.EXE symbol vector that resides in another image.

In general, at any one point in time, the debugger can access only the symbols from one image. It does this to reduce the time it takes to search for a symbol in a table. To load the symbols for a particular image, use the SET IMAGE command. When you set an image, the debugger loads all the symbols from the new image and makes that image the current image. The symbols from the previous image are in memory, but the debugger will not look through
them to translate symbols. To remove symbols from memory for an image, use the CANCEL IMAGE command (which does not work on the main image, SYS$BASE_IMAGE).

There is a set of modules for each image the debugger accesses. The symbol tables in the image that are part of these modules are not loaded with the SET IMAGE command. Instead they can be loaded with the SET MODULE <module-name> or SET MODULE/ALL commands. As they are loaded, a new symbol table is created in memory under the symbol table for the image. Figure 8–1 shows what this looks like.

Figure 8–1 Maintaining Symbols

![Symbol Table Diagram]

When the debugger needs to look up a symbol name, it first looks at the current image to find the information. If it does not find it there, it then looks into the appropriate module. It determines which module is appropriate by looking at the module range symbols which are part of the image symbol table.

To see the symbols that are currently loaded, use the debugger’s SHOW SYMBOL command. This command has a few options to obtain more than just the symbol name and value. (See the OpenVMS Debugger Manual for more details.)

8.11.2 Overview of OpenVMS Executive Image Symbols

Depending on whether the debugger has access to the .DSF or .EXE file, different kinds of symbols could be loaded. Most users will have the .EXE file for the OpenVMS executive images and a .DSF file for their private images—that is, the images they are debugging.

The OpenVMS executive consists of two base images, SYS$BASE_IMAGE.EXE and SYS$PUBLIC_VECTORS.EXE, and a number of separately loadable executive images.

The two base images contain symbol vectors. For SYS$BASE_IMAGE.EXE, the symbol vector is used to define symbols accessible by all the separately loadable images. This allows these images to communicate with each other through cross-image routine calls and memory references. For SYS$PUBLIC_VECTORS.EXE, the symbol vector is used to define the OpenVMS system services. Because these symbol vectors are in the .EXE and the .DSF files, the debugger can load these symbols no matter which one you have.

All images in the OpenVMS executive also contain global and local symbols. However, none of these symbols ever gets into the .EXE file for the image. These symbols are put in the specific module’s section of the .DSF file if that module was compiled using /DEBUG and the image was linked using /DSF.
8.11.3 Possible Problems You May Encounter

- **Access to All Executive Image Symbols**
  
  When the current image is not SYS$BASE_IMAGE, but one of the separately loaded images, the debugger does not have access to any of the symbols in the SYS$BASE_IMAGE symbol vector. This means you cannot access (set breakpoints, and so on) any of the cross-image routines or data cells. The only symbols you have access to are the ones defined by the current image.

  If the debugger has access only to the .EXE file, then only symbols that have vectors in the base image are accessible. For .DSF files, the current image symbols for any set module are defined. (You can tell if you have the .DSF or .EXE by using the SHOW MODULE command—if there are no modules you have the .EXE). This includes any symbols in the SYS$BASE_IMAGE.EXE symbol vector for which the code or data resides in the current image.

  However, the user cannot access a symbol that is part of the SYS$BASE_IMAGE.EXE symbol vector that resides in another image. For example, if you are in one image and you want to set a breakpoint in a cross-image routine from another image, you do not have access to the symbol. Of course, if you know in which image it is defined, you can do a SET IMAGE, SET MODULE/ALL, and then a SET BREAK.

  There is a debugger workaround for this problem. The debugger and SCD let you use the SET MODULE command on an image by prefixing the image name with SHARE$ (SHARE$SYS$BASE_IMAGE, for example). This treats that image as a module which is part of the current image. In the previous figure, think of it as another module in the module list for an image. Note, however, that only the symbols for the symbol vector are loaded. None of the symbols for the modules of the SHARE$xxx image are loaded. Therefore, this command is only useful for base images.

  So, in other words, by doing SET MODULE SHARE$SYS$BASE_IMAGE, the debugger gives you access to all cross-image symbols for the OpenVMS executive.

- **Stale Data from the Symbol Vector**

  When an OpenVMS executive based image is loaded, the values in the symbol vectors are only correct for information that resides in that based image. For all symbols that are defined in the separately loaded images, the based image contains a pointer to a placeholder location. For routine symbols this is a routine that just returns "an image not loaded" failure code. A symbol vector entry is fixed to contain the real symbol address when the image in which the data resides is loaded.

  Therefore, if you do a SET IMAGE command to a base image before all the symbol entries are corrected, the SET IMAGE obtains the placeholder value for those symbols. Then, once the image containing the real data is loaded, the debugger will still have the placeholder value. This means that you are looking at stale data. One solution to this is to make sure to do a SET IMAGE command on the base image in order to get the most up-to-date symbol vector loaded into memory.

  The CANCEL IMAGE/SET IMAGE combination does not currently work for SYS$BASE_IMAGE because it is the main image and DEBUG does not allow you to CANCEL the main image. Therefore, if you connect to the target system early in the boot process, you will have stale data as part of the SYS$BASE_IMAGE symbol table. However, the SET MODULE SHARE$xxx command always reloads the information from the symbol vector. So, to solve
this problem you could SET IMAGE to an image other than SYS$BASE_IMAGE and then use the CANCEL MODULE SHARE$SYS$BASE_IMAGE and SET MODULE SHARE$SYS$BASE_IMAGE commands to do the same thing. The only other solution is to always connect to the target system once all images are loaded that define the real data for values in the symbol vectors. You could also enter the following commands, and you would obtain the latest values from the symbol vector:

SET IMAGE EXEC_INIT
SET MODULE/ALL
SET MODULE SHARE$SYS$BASE_IMAGE

• Problems with SYS$BASE_IMAGE.DSF

For those who have access to the SYS$BASE_IMAGE.DSF file, there may be another complication with accessing symbols from the symbol vector. The problem is that the module SYSTEM_ROUTINES contains the placeholder values for each symbol in the symbol vector. So, if SYSTEM_ROUTINES is the currently set module (which is the case if you are sitting at the INI$BRK breakpoint) then the debugger will have the placeholder value of the symbol as well as the value in the symbol vector. You can see what values are loaded with the SHOW SYMBOL/ADDRESS command. The symbol vector version should be marked with (global); the local one is not.

To set a breakpoint at the correct code address for a routine when in this state, use the SHOW SYMBOL/ADDRESS command on the routine symbol name. If the global and local values for the code address are the same, then the image with the routine has not yet been loaded. If not, set a breakpoint at the code address for the global symbol.
8.12 Sample System Code Debugging Session

This section provides a sample session that shows the use of some OpenVMS debugger commands as they apply to SCD. The examples in this session show how to work with C code that has been linked into the SYSTEM_DEBUG execlet. It is called as an initialization routine for SYSTEM_DEBUG.

To reproduce this sample session, the host system needs access to the SYSTEM_DEBUG.DSF matching the SYSTEM_DEBUG.EXE file on your target system, and to the source file C_TEST_ROUTINES.C, which is available in SYS$EXAMPLES. The target system is booted with the boot flags 0, 8004, so it stops at an initial breakpoint, and the devices DKB200,ESA0.

Example 8–1 Booting the Target System

```
>>> b -f1 0,8004 dkb200,esa0
INIT-S-CPU...
INIT-S-RESET_TC...
INIT-S-ASIC...
INIT-S-MEM...
INIT-S-NVR...
INIT-S-SCC...
INIT-S-NI...
INIT-S-SCSI...
INIT-S-ISDN...
INIT-S-TC0...
AUDIT_BOOT_STARTS ...
AUDIT_CHECKSUM_GOOD
AUDIT_LOAD_BEGINS
AUDIT_LOAD_DONE
%SYSBOOT-I-GCTFIL, Using a configuration file to boot as a Galaxy instance.

OpenVMS (TM) Alpha Operating System, Version V7.2

DBGTK: Initialization succeeded. Remote system debugging is now possible.

DBGTK: Waiting at breakpoint for connection from remote host.

The example continues by invoking the system code debugger’s character cell interface on the host system.

Example 8–2 Invoking the System Code Debugger

```
$ define dbg$decw$display " "
$ debug/keep

OpenVMS Alpha Debug64 Version V7.2-019

DBG>
```
Use the CONNECT command to connect to the target system. In this example, the target system's default password is the null string, and the logical name DBGHK$IMAGE_PATH is used for the image path; so the command qualifiers /PASSWORD and /IMAGE_PATH are not being used. You may need to use them.

When you have connected to the target system, the DEBUG prompt is displayed. Enter the SHOW IMAGE command to see what has been loaded. Because you are reaching a breakpoint early in the boot process, there are very few images. See Example 8-3. Notice that SYS$BASE_IMAGE has an asterisk next to it. This is the currently set image, and all symbols currently loaded in the debugger come from that image.

### Example 8–3 Connecting to the Target System

```plaintext
DBG> connect %node_name TSTSYS
%DEBUG-I-INIBRK, target system interrupted
%DEBUG-I-DYNMODSET, setting module SYSTEM_ROUTINES

DBG> show image

<table>
<thead>
<tr>
<th>image name</th>
<th>set</th>
<th>base address</th>
<th>end address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRORLOG</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80808400</td>
<td>FFFFFFFF808086FF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80CA3600</td>
<td>FFFFFFFF80CA3BFF</td>
</tr>
<tr>
<td>EXEC_INIT</td>
<td>no</td>
<td>FFFFFFFF8306E000</td>
<td>FFFFFFFF830A2000</td>
</tr>
<tr>
<td>*SYS$BASE_IMAGE</td>
<td>yes</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80002000</td>
<td>FFFFFFFF8000EDFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C05C00</td>
<td>FFFFFFFF80C2AFFF</td>
</tr>
<tr>
<td>SYS$CNBTDRCVR</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF8001A000</td>
<td>FFFFFFFF8001AFFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C2600</td>
<td>FFFFFFFF80C2D9FF</td>
</tr>
<tr>
<td>SYS$CPU_ROUTINES_0402</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80010000</td>
<td>FFFFFFFF800191FF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C2B000</td>
<td>FFFFFFFF80C2DFFF</td>
</tr>
<tr>
<td>SYS$ESBTDRIVER</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80C30C00</td>
<td>FFFFFFFF80C30FFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C80000</td>
<td>FFFFFFFF80C80FFF</td>
</tr>
<tr>
<td>SYS$NISCU_BTDRCVR</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF8001C000</td>
<td>FFFFFFFF8002ADF1FF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C2D000</td>
<td>FFFFFFFF80C2DF1FF</td>
</tr>
<tr>
<td>SYS$SMTDRIVER</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80030000</td>
<td>FFFFFFFF80037FFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C31000</td>
<td>FFFFFFFF80C31FFF</td>
</tr>
<tr>
<td>SYS$PUBLIC_VECTORS</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80000000</td>
<td>FFFFFFFF80001FFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C00000</td>
<td>FFFFFFFF80C05BFFF</td>
</tr>
<tr>
<td>SYSTEM_DEBUG</td>
<td>no</td>
<td>FFFFFFFF802FE8000</td>
<td>FFFFFFFF80356000</td>
</tr>
<tr>
<td>SYSTEM_PRIMITIVES_MIN</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80034000</td>
<td>FFFFFFFF80077FFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80C31000</td>
<td>FFFFFFFF80C311FF</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION_UNI</td>
<td>no</td>
<td>0000000000000000</td>
<td>FFFFFFFFEFFFFF</td>
</tr>
<tr>
<td>NPRO0</td>
<td></td>
<td>FFFFFFFF80780000</td>
<td>FFFFFFFF80835FFF</td>
</tr>
<tr>
<td>NPRW1</td>
<td></td>
<td>FFFFFFFF80CA1200</td>
<td>FFFFFFFF80CA35FFF</td>
</tr>
</tbody>
</table>

total images: 12 bytes allocated: 1517736
Example 8–4 Target System Connection Display

DBGTK: Connection attempt from host HSTSYS user GUEST process 2E801C2F
DBGTK: Connection attempt succeeded

To set a breakpoint at the first routine in the C_TEST_ROUTINES module of the
SYSTEM_DEBUG.EXE execlet, do the following:

1. Load the symbols for the SYSTEM_DEBUG image with the DEBUG SET
   IMAGE command.
2. Use the SET MODULE command to obtain the symbols for the module.
3. Set the language to be C and set a breakpoint at the routine test_c_code.
   The language must be set because C is case sensitive and test_c_code needs
   to be specified in lowercase. The language is normally set to the language of
   the main image, in this example SYS$BASE_IMAGE.EXE. Currently that is
   not C.

Example 8–5 Setting a Breakpoint

DBG> set image system_debug
DBG> show module

module name symbols language size
AUX_TARGET no C 15928
BUFSRV_TARGET no C 11288
BUGCHECK_CODES no BLISS 26064
CRTLPRINTF no C 29920
C_TEST_ROUTINES no C 3808
FATAL_EXC no C 1592
HIGH_ADDRESS no C 372
LIB$CALLING_STANDARD_AUX no MACRO64 1680
LINMGR_TARGET no C 13320
LOW_ADDRESS no C 368
OBJMGR no C 5040
PLUMGR no C 19796
POOL no C 116
PROTOMGR_TARGET no C 17868
SOCMGR no C 3324
SYS$DOINIT no AMACRO 81740
TARGET_KERNEL no C 207244
TMRMGR_TARGET no C 3516
XDELTA no BLISS 189940
XDELTA_ISR no MACRO64 2428

total modules: 20. bytes allocated: 1585168.
Example 8–5 (Cont.) Setting a Breakpoint

```
DBG> set module c_test_routines
DBG> show module c_test_routines
module name  symbols  size
C_TEST_ROUTINES  yes  3808

Now that the breakpoint is set, you can proceed and activate the breakpoint. When that occurs, the debugger tries to open the source code for that location in the same place as where the module was compiled. Because that is not the same place as on your system, you need to tell the debugger where to find the source code. This is done with the debugger's SET SOURCE command, which takes a search list as a parameter so you can make it point to many places.
```

Example 8–6 Finding the Source Code

```
DBG> set source/latest sys$examples,sys$library
DBG> go
break at routine C_TEST_ROUTINES\test_c_code
  166: x = xdt$fregsav[0];
```
Now that the debugger has access to the source, you can put the debugger into screen mode to see exactly where you are and the code surrounding it.

**Example 8–7 Using the Set Mode Screen Command**

```
DBG> Set Mode Screen; Set Step Nosource
- SRC: module C_TEST_ROUTINES -scroll-source----------------------------------
  151:  xdt$fregsav[5] = in64;
  153:  if (xdt$fregsav[9] > 0)
  154:      *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
  155:  else
  156:      *pVar = (*pVar + xdt$fregsav[17]);
  157:  xdt$fregsav[7] = test_c_code3(10);
  158:  xdt$fregsav[3] = test;
  159:  return xdt$fregsav[23];
  160: }
  161: void test_c_code(void)
  162: {
  163:    int x, y;
  164:    int64 x64, y64;
  165:    x = xdt$fregsav[0];
  166:    y = xdt$fregsav[1];
  167:    x64 = xdt$fregsav[2];
  168:    y64 = xdt$fregsav[3];
  169:    xdt$fregsav[14] = test_c_code2(x64+y64, x+y, x64+x, &y64);
  170:    test_c_code4();
  171:    return;
  172: }
- OUT -output-------------------------------------------------------------------

- PROMPT -error-program-prompt--------------------------------------------------

DBG>
```
Now, you want to set another breakpoint inside the test_c_code3 routine. You use the debugger’s SCROLL/UP command (8 on the keypad) to move to that routine and see that line 146 would be a good place to set the breakpoint. It is at a recursive call. Then you proceed to that breakpoint with the GO command.

Example 8–8 Using the SCROLL/UP DEBUG Command

```
- SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
  133: void test_c_code4(void)
  134: {
  135:    int i,k;
  136:    for(k=0;k<1000;k++)
  137:      {
  138:        test_c_code5(&i);
  139:      }
  140:    return;
  141: }
  142: int test_c_code3(int subrtnCount)
  143: {
  144:    subrtnCount = subrtnCount - 1;
  145:    if (subrtnCount != 0)
  146:      subrtnCount = test_c_code3(subrtnCount);
  147:    return subrtnCount;
  148: }
  149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
  150: {
  151:    xdt$fregsav[5] = in64;
  153:    if (xdt$fregsav[9] > 0)
  154:      *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
  155:    else
- OUT -output-------------------------------------------------------------------
- PROMPT -error-program-prompt--------------------------------------------------
```

DBG> Scroll/Up
DBG> set break %line 146
DBG> go
DBG>
When you reach that breakpoint, the source code display is updated to show where you currently are, which is indicated by an arrow. A message also appears in the OUT display indicating you reach the breakpoint at that line.

Example 8–9  Breakpoint Display

```
-SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
 135:   int i,k;
 136:   for(k=0;k<1000;k++)
 137:     {
 138:       test_c_code5(&i);
 139:     }
 140:   return;
 141: }
 142: int test_c_code3(int subrtnCount)
 143: {
 144:   subrtnCount = subrtnCount - 1;
 145:   if (subrtnCount != 0)
-> 146:     subrtnCount = test_c_code3(subrtnCount);
 147:   return subrtnCount;
 148: }
 149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
 150: {
 151:   xdt$fregsav[5] = in64;
 153:   if (xdt$fregsav[9] > 0)
 154:     *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
 155:   else
 156:     *pVar = (*pVar + xdt$fregsav[17]);
 157:   xdt$fregsav[7] = test_c_code3(10);
-OUT -output-------------------------------------------------------------------
break at C_TEST_ROUTINES\test_c_code3\%LINE 146

-PROMPT -error-program-prompt--------------------------------------------------
DBG> Scroll/Up
DBG> set break %line 146
DBG> go
DBG>
```
Now you try the debugger’s STEP command. The default behavior for STEP is STEP/OVER, unlike XDELTA and DELTA, which is STEP/INTO, so, normally you would expect to step to line 147 in the code. However, because you have a breakpoint inside test_c_code3 that is called at line 146, you will reach that event first.

**Example 8-10 Using the Debug Step Command**

```plaintext
- SRC: module C_TEST_ROUTINES -scroll-source-------------------------------
  135:   int i,k;
  136:   for(k=0;k<1000;k++)
  137:   {
  138:     test_c_code5(&i);
  139:   }
  140:   return;
  141: }
  142: int test_c_code3(int subrtnCount)
  143: {
  144:   subrtnCount = subrtnCount - 1;
  145:   if (subrtnCount != 0)
  146:   -> subrtnCount = test_c_code3(subrtnCount);
  147:   return subrtnCount;
  148: }
  149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
  150: {
  151:   xdt$fregsav[5] = in64;
  153:   if (xdt$fregsav[9] > 0)
  154:     *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
  155:   else
  156:     *pVar = (*pVar + xdt$fregsav[17]);
  157:   xdt$fregsav[7] = test_c_code3(10);
- OUT -output---------------------------------------------------------------
break at C_TEST_ROUTINES\test_c_code3\%LINE 146
break at C_TEST_ROUTINES\test_c_code3\%LINE 146

- PROMPT -error-program-prompt-----------------------------------------------
DBG>
DBG> set break %line 146
DBG> go
DBG> Step
DBG>
```
Now, you try a couple of other commands, EXAMINE and SHOW CALLS. The EXAMINE command allows you to look at all the C variables. Note that the C_TEST_ROUTINES module is compiled with the /NOOPTIMIZE switch which allows access to all variables. The SHOW CALLS command shows you the call sequence from the beginning of the stack. In this case, you started out in the image EXEC_INIT. (The debugger prefixes all images other than the main image with SHARE$ so it shows up as SHARE$EXEC_INIT.)

Example 8–11 Using the Examine and Show Calls Commands

```c
- SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
  135:     int i,k;
  136:     for(k=0;k<1000;k++)
  137:     {
  138:         test_c_code5(&i);
  139:     }
  140:     return;
  141: }
  142: int test_c_code3(int subrtnCount)
  143: {
  144:     subrtnCount = subrtnCount - 1;
  145:     if (subrtnCount != 0)
  146:         subrtnCount = test_c_code3(subrtnCount);
  147:     return subrtnCount;
  148: }
  149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
  150: {
  151:     xdt$fregsav[5] = in64;
  153:     if (xdt$fregsav[9] > 0)
  154:         *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
  155:     else
  156:         *pVar = (*pVar + xdt$fregsav[17]);
  157:     xdt$fregsav[7] = test_c_code3(10);
- OUT -output-------------------------------------------------------------------
break at C_TEST_ROUTINES\test_c_code3\%LINE 146
break at C_TEST_ROUTINES\test_c_code3\%LINE 146
C_TEST_ROUTINES\test_c_code3\subrtnCount: 8
module name routine name line rel PC abs PC
*C_TEST_ROUTINES test_c_code3 146 00000000000000C4 FFFFFFFF83002D64
*C_TEST_ROUTINES test_c_code3 146 00000000000000D4 FFFFFFFF83002D74
*C_TEST_ROUTINES test_c_code3 146 00000000000001A0 FFFFFFFF83002E40
*C_TEST_ROUTINES test_c_code3 146 0000000000000260 FFFFFFFF83002F00
*XDELTA xdt$SYSDBG_INIT 9371 00000000000005B8 FFFFFFFF83052238
*SYS$DOINIT INI$DOINIT 1488 000000000000098 FFFFFFFF83052B8
SHARE$EXEC_INIT 00000000000018C74 FFFFFFFF83086C74
SHARE$EXEC_INIT 00000000000014BD0 FFFFFFFF83082BD0
- PROMPT -error-program-prompt--------------------------------------------------
DBG>
DBG> set break %line 146
DBG> go
DBG> Step
DBG> examine subrtnCount
DBG> show calls
DBG>
```
If you want to proceed because you are done debugging this code, first cancel all the breakpoints and then enter the GO command. Notice, however, that you do not keep running but receive a message that you have stepped to line 147. This happens because the STEP command used earlier never completed. It was interrupted by the breakpoint on line 146.

Note that the debugger remembers all step events and only removes them once they have completed.

Example 8–12  Canceling the Breakpoints

---

- SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
  136: for(k=0;k<1000;k++)
  137: {
  138:     test_c_code5(&i);
  139: }
  140: return;

  142: int test_c_code3(int subrtnCount)
  143: {
  144:     subrtnCount = subrtnCount - 1;
  145:     if (subrtnCount != 0)
  146:         subrtnCount = test_c_code3(subrtnCount);
  147: return subrtnCount;

  149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
  150: {
  151:     xdt$fregsav[5] = in64;
  153:     if (xdt$fregsav[9] > 0)
  154:         *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
  155:     else
  156:         *pVar = (*pVar + xdt$fregsav[17]);
  157:     xdt$fregsav[7] = test_c_code3(10);
  158:     xdt$fregsav[3] = test;

- OUT -output-------------------------------------------------------------------
break at C_TEST_ROUTINES\test_c_code3\%LINE 146
break at C_TEST_ROUTINES\test_c_code3\%LINE 146

 C_TEST_ROUTINES\test_c_code3\subrtnCount:   8

module name routine name line rel PC abs PC
*C_TEST_ROUTINES test_c_code3 146 00000000000000C4 FFFFFFFF83002D64
*C_TEST_ROUTINES test_c_code2 146 00000000000000D4 FFFFFFFF83002D74
*C_TEST_ROUTINES test_c_code2 157 00000000000001A0 FFFFFFFF83002E40
*C_TEST_ROUTINES test_c_code 170 000000000000260 FFFFFFFF83002F00
*XDELTA XDT$SYSDBG_INIT 9371 000000000000058 FFFFFFFF83052238
*SYS$DOINIT INI$DOINIT 1488 00000000000098 FFFFFFFF83052B88
SHARE$EXEC_INIT 000000000018C74 FFFFFFFF83086C74
SHARE$EXEC_INIT 000000000014BD0 FFFFFFFF83082BD0
stepped to C_TEST_ROUTINES\test_c_code3\%LINE 147

- PROMPT -error-program-prompt--------------------------------------------------
  DBG> go
  DBG> Step
  DBG> examine subrtnCount
  DBG> show calls
  DBG> cancel break/all
  DBG> go
  DBG>
The OpenVMS Alpha System Code Debugger
8.12 Sample System Code Debugging Session

The STEP/RETURN command, a different type of step command, single steps assembly code until it finds a return instruction. This command is useful if you want to see the return value for the routine, which is done here by examining the R0 register.

For more information about using other STEP command qualifiers, see the OpenVMS Debugger Manual.

Example 8–13 Using the Step/Return Command

```c
- SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
  137: { test_c_code5(&i);
  139: }
  140: return;
  141: }
  142: int test_c_code3(int subrtnCount)
  143: {
  144:     subrtnCount = subrtnCount - 1;
  145:     if (subrtnCount != 0)
  146:     subrtnCount = test_c_code3(subrtnCount);
  147:     return subrtnCount;
  -> 148: }
  149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
  150: {
  151:     xdt$fregsav[5] = in64;
  153:     if (xdt$fregsav[9] > 0)
  154:         *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
  155:     else
  156:         *pVar = (*pVar + xdt$fregsav[17]);
  157:     xdt$fregsav[7] = test_c_code3(10);
  158:     xdt$fregsav[3] = test;
  159:     return xdt$fregsav[23];
- OUT -output-------------------------------------------------------------------
brea at C_TEST_ROUTINES\test_c_code3\%LINE 146
brea at C_TEST_ROUTINES\test_c_code3\%LINE 146
C_TEST_ROUTINES\test_c_code3\subrtnCount:  8
module name routine name line rel PC abs PC
*C_TEST_ROUTINES test_c_code3 146 00000000000000c4 FFFFFFFF83002D64
*C_TEST_ROUTINES test_c_code3 146 00000000000000d4 FFFFFFFF83002D74
*C_TEST_ROUTINES test_c_code2 157 00000000000001a0 FFFFFFFF83002E40
*C_TEST_ROUTINES test_c_code 170 0000000000000260 FFFFFFFF83002F00
*XDELTA XTSSYSDBG_INIT 9371 0000000000000058 FFFFFFFF83052238
*SYS$DOINIT INISDOINIT 1488 0000000000000098 FFFFFFFF830520B8
SHARE$EXEC_INIT 0000000000018C74 FFFFFFFF83086C74
SHARE$EXEC_INIT 0000000000014BD0 FFFFFFFF83082BD0
stepped to C_TEST_ROUTINES\test_c_code3\%LINE 147
stepped on return from C_TEST_ROUTINES\test_c_code3\%LINE 147 to C_TEST_ROUTINES\test_c_code3\%LINE 148
C_TEST_ROUTINES\test_c_code3\%R0: 0
- PROMPT -error-program-prompt-----------------------------------------------
  DBG> examine subrtnCount
  DBG> show calls
  DBG> cancel break/all
  DBG> go
  DBG> step/return
  DBG> examine r0
  DBG>
```
After you finish the SCD session, enter the GO command to leave this module. You will encounter another INI$BRK breakpoint at the end of EXEC_INIT. An error message indicating there are no source lines for address 80002010 is displayed, because debug information on this image or module is not available. Also notice that there is no message in the OUT display for this event. That is because INI$BRKs are special breakpoints that are handled as SS$_DEBUG signals. They are a method for the system code to break into the debugger and there is no real breakpoint in the code.

Example 8–14 Source Lines Error Message

```
- SRC: module SYSTEM_ROUTINES -scroll-source------------------------------------
15896: Source line not available
15897: Source line not available

15906: Source line not available
-5907: Source line not available
15908: Source line not available

15917: Source line not available
15918: Source line not available

- OUT -output-------------------------------------------------------------------
break at C_TEST_ROUTINES\test_c_code3\%LINE 146
break at C_TEST_ROUTINES\test_c_code3\%LINE 146
C_TEST_ROUTINES\test_c_code3\subrtnCount: 8

module name    routine name    line rel PC        abs PC
*C_TEST_ROUTINES test_c_code3 146 00000000000000C4 FFFFFFFF83002D64
*C_TEST_ROUTINES test_c_code3 146 00000000000000D4 FFFFFFFF83002D74
*C_TEST_ROUTINES test_c_code2 157 00000000000001A0 FFFFFFFF83002E40
*C_TEST_ROUTINES test_c_code 170 0000000000000260 FFFFFFFF83002F00
*XDELT XDS$SYSDBG_INIT 9371 0000000000000058 FFFFFFFF83052238
*SYS$DOINIT INIS$DOINIT 1488 0000000000000098 FFFFFFFF830520B8
SHARE$EXEC_INIT 000000000000018C74 FFFFFFFF83086C74
SHARE$EXEC_INIT 000000000000014BD0 FFFFFFFF83082BD0
stepped to C_TEST_ROUTINES\test_c_code3\%LINE 147
stepped on return from C_TEST_ROUTINES\test_c_code3\%LINE 147 to C_TEST_ROUTINES\test_c_code3\%LINE 148
C_TEST_ROUTINES\test_c_code3\%R0: 0
- PROMPT -error-program-prompt------------------------------------------------
DBG> examine r0
DBG> go
%DEBUG-I-INIBRK, target system interrupted
%DEBUG-I-DYNMIGSET, setting image SYS$BASE_IMAGE
%DEBUG-W-SCRUNAOPNSRC, unable to open source file SYS$COMMON:[SYSLIB]SYSTEM_ROUTINES.M64;
-RMS-E-FNF, file not found
DBG>
```
Enter the SHOW IMAGE command. You will see more images displayed as the boot path has progressed further.

Finally, enter GO, allowing the target system to boot completely, because there are no more breakpoints in the boot path. The debugger will wait for another event to occur.

Example 8–15 Using the Show Image Command

```
- SRC: module SYSTEM_ROUTINES -scroll-source------------------------------------
15896: Source line not available
15897: Source line not available
 .
 .
15906: Source line not available
->5907: Source line not available
15908: Source line not available
 .
 .
15917: Source line not available
15918: Source line not available
- OUT -output-------------------------------------------------------------------
PRG2 FFFFFFFF8329C000 FFFFFFFF832A2DFF
SYSLICENSE no 0000000000000000 FFFFFFFFFFFFFFFF
NPW0 FFFFFFFF80188000 FFFFFFFF801883FF
NPW1 FFFFFFFF800CC5000 FFFFFFFF800CC5FF
PRG2 FFFFFFFF83221E000 FFFFFFFF832247FF
PRW3 FFFFFFFF83226000 FFFFFFFF832265FF
SYSTEM_DEBUG yes FFFFFFFF82FFE000 FFFFFFFF83056000
SYSTEM_PRIMITIVES_MIN no 0000000000000000 FFFFFFFFFFFFFFFF
NPW0 FFFFFFFF80034000 FFFFFFFF800775FF
NPW1 FFFFFFFF800C31A00 FFFFFFFFFF80CA11FF
SYSTEM_SYNCHRONIZATION_UNI no 0000000000000000 FFFFFFFFFFFFFFFF
NPW0 FFFFFFFF80078000 FFFFFFFF800835FF
NPW1 FFFFFFFF800CA1200 FFFFFFFFFF80CA35FF
total images: 40 bytes allocated: 2803296
- PROMPT -error-program-prompt--------------------------------------------------
%DEBUG-I-INIBRK, target system interrupted
%DEBUG-I-DYNIMGSET, setting image SYS$BASE_IMAGE
%DEBUG-W-SCRUNAOPNSRC, unable to open source file X6P3_RESD$:\[SYSLIB\]SYSTEM_ROUTINES.M64;
-RMS-E-FNF, file not found
DBG> show image
DBG> go
```
This chapter describes the OpenVMS Alpha System Dump Debugger (SDD) and how you can use it to analyze system crash dumps.

SDD is similar in concept to SCD as described in Chapter 8. Where SCD allows connection to a running system with control of the system's execution and the examination and modification of variables, SDD allows analysis of memory as recorded in a system dump.

Use of the SDD usually involves two systems, although all the required environment can be set up on a single system. The description that follows assumes that two systems are being used:

• The build system, where the image that causes the system crash has been built

• The test system, where the image is executed and the system crash occurs

In common with SCD, the OpenVMS debugger's user interface allows you to specify variable names, routine names, and so on, precisely as they appear in your source code. Also, SDD can display the source code where the software was executing at the time of the system crash.

SDD recognizes the syntax, data typing, operators, expressions, scoping rules, and other constructs of a given language. If your code or driver is written in more than one language, you can change the debugging context from one language to another during a debugging session.

To use SDD, you must do the following:

• Build the system image or device driver that is causing the system crash.

• Boot a system, including the system image or device driver, and perform the necessary steps to cause the system crash.

• Reboot the system and save the dump file.

• Invoke SDD, which is integrated with the OpenVMS debugger.

The following sections cover these tasks in more detail, describe the available user-interface options, summarize applicable OpenVMS Debugger commands, and provide a sample SDD session.

9.1 User-Interface Options

SDD has the following user-interface options.

• A DECwindows Motif interface for workstations.

   When using this interface, you interact with SDD by using a mouse and pointer to choose items from menus, click on buttons, select names in windows, and so on.
9.1 User-Interface Options

Note that you can also use OpenVMS Debugger commands with the DECwindows Motif interface.

• A character cell interface for terminals and workstations.
When using this interface, you interact with SDD by entering commands at a prompt. The sections in this chapter describe how to use the system dump debugger with the character cell interface.
For more information about using the OpenVMS DECwindows Motif interface and OpenVMS Debugger commands with SDD, see the OpenVMS Debugger Manual.

9.2 Preparing a System Dump to Be Analyzed

To prepare a system dump for analysis, perform the following steps:

1. Compile the sources you will want to analyze, and use the /DEBUG (mandatory) and /NOOPT (preferred) qualifiers.

   Note
Because you are analyzing a snapshot of the system, it is not as vital to use unoptimized code as it is with the system code debugger. But note that you cannot access all variables. SDD may report that they are optimized away.

2. Link your image using the /DSF (debug symbol file) qualifier. Do not use the /DEBUG qualifier, which is for debugging user programs. The /DSF qualifier takes an optional filename argument similar to the /EXE qualifier. For more information, see the OpenVMS Linker Utility Manual. If you specify a name in the /EXE qualifier, you will need to specify the same name for the /DSF qualifier. For example, you would use the following command:

   $ LINK/EXE=EXE$:MY_EXECLET/DSF=EXE$:MY_EXECLET OPTIONS_FILE/OPT

   The .DSF and .EXE file names must be the same. Only the extensions will be different, that is, .DSF and .EXE.

   The contents of the .EXE file should be exactly the same as if you had linked without the /DSF qualifier. The .DSF file will contain the image header and all the debug symbol tables for .EXE file. It is not an executable file, and cannot be run or loaded.

3. Put the .EXE file on your test system.

4. Boot the test system and perform the necessary steps to cause the system crash.

5. Reboot the test system and copy the dump to the build system using the System Dump Analyzer (SDA) command COPY. See Chapter 4.
9.3 Setting Up the Test System

The only requirement for the test system is that the .DSF file matching the .EXE file that causes the crash is available on the build system.

There are no other steps necessary in the setup of the test system. With the system image copied to the test system, it can be booted in any way necessary to produce the system crash. Since SDD can analyze most system crash dumps, any system can be used, from a standalone system to a member of a production cluster.

Note

It is assumed that the test system has a dump file large enough for the system dump to be recorded. Any dump style may be used (full or selective, compressed or uncompressed). A properly AUTOGENed system will meet these requirements.

9.4 Setting Up the Build System

To set up the build system, you need access to all system images and drivers that were loaded on the test system. You should have access to a source listings kit or a copy of the following directories:

SYS$LOADABLE_IMAGES:
SYS$LIBRARY:
SYS$MESSAGE:

You need all the .EXE files in those directories. The .DSF files are available with the OpenVMS Alpha source listings kit.

 Optionally, you need access to the source files for the images to be debugged. SDD will look for the source files in the directory where they were compiled. You must use the SET SOURCE command to point SDD to the location of the source code files if they are not in the directories used when the image was built. For an example of the SET SOURCE command, see Section 9.9.

Before you can analyze a system dump with SDD, you must set up the logical name DBGHK$IMAGE_PATH, which must be set up as a search list to the area where the system images or .DSF files are kept. For example, if the copies are in the following directories:

DEVICE:[SYS$LDR]
DEVICE:[SYSLIB]
DEVICE:[SYSMSG]

you would define DBGHK$IMAGE_PATH as follows:

$ define dbghk$image_path DEVICE:[SYS$LDR],DEVICE:[SYSLIB],DEVICE:[SYSMSG]

This works well for analyzing a system dump using all the images normally loaded on a given system. However, you might be using SDD to analyze new code either in an execlet or a new driver. Because that image is most likely in your default directory, you must define the logical name as follows:

$ define dbghk$image_path [],DEVICE:[SYS$LDR],DEVICE:[SYSLIB],DEVICE:[SYSMSG]
If SDD cannot find one of the images through this search path, a warning message is displayed. SDD will continue initialization as long as it finds at least one image. If SDD cannot find the SYS$BASE_IMAGE file, which is the OpenVMS Alpha operating system's main image file, an error message is displayed and the debugger exits.

If and when this happens, check the directory for the image files and compare it to what was loaded on the test system.

9.5 Starting the System Dump Debugger

To start SDD on the build system, enter the following command.

```
$ DEBUG/KEEP
```

SDD displays the DBG> prompt. With the DBGHK$IMAGE_PATH logical name defined, you can invoke the ANALYZE/CRASH_DUMP command and optional qualifier /IMAGE_PATH.

To use the ANALYZE/CRASH_DUMP command and optional qualifier (/IMAGE_PATH) to analyze the dump in file <file-name> enter the following command:

```
DBG> ANALYZE/CRASH_DUMP file-name
```

The /IMAGE_PATH qualifier is optional. If you do not use this qualifier, SDD uses the DBGHK$IMAGE_PATH logical name as the default. The /IMAGE_PATH qualifier is a quick way to change the logical name. However, when you use it, you cannot specify a search list. You can use only a logical name or a device and directory, although the logical name can be a search list.

Usually, SDD obtains the source file name from the object file. This is put there by the compiler when the source is compiled with the /DEBUG qualifier. The SET SOURCE command can take a list of paths as a parameter. It treats them as a search list.

9.6 Summary of System Dump Debugger Commands

Only a subset of OpenVMS debugger commands can be used in SDD. The following are a few examples of commands that you can use in SDD:

- Commands to manipulate the source display, such as TYPE and SCROLL
- Commands used in OpenVMS debugger command programs, such as DO and IF
- Commands that affect output formats, such as SET RADIX
- Commands that manipulate symbols and scope, such as EVALUATE, SET LANGUAGE, and CANCEL SCOPE
- Commands that read the contents of memory and registers, such as EXAMINE

Examples of commands that cannot be used in SDD are as follows:

- Commands that cause code to be executed, such as STEP and GO
- Commands that manipulate breakpoints, such as SET BREAK and CANCEL BREAK
- Commands that modify memory or registers, such as DEPOSIT
You can also use the OpenVMS debugger command SDA to examine the system dump with System Dump Analyzer semantics. This command, which is not available when debugging user programs, is described in the next section.

9.7 Using System Dump Analyzer Commands

Once a dump file has been opened, you can use the commands listed in the previous section to examine the system dump. You can also use some System Dump Analyzer (SDA) commands, such as SHOW SUMMARY and SHOW DEVICE. This feature allows the system programmer to take advantage of the strengths of both the OpenVMS Debugger and SDA to examine the system dump and to debug system programs such as device drivers, without having to invoke both the OpenVMS debugger and SDA separately.

To obtain access to SDA commands, you simply type "SDA" at the OpenVMS Debugger prompt ("DBG>") at any time after the dump file has been opened. SDA initializes itself and then outputs the "SDA>" prompt. Enter SDA commands as required. (See Chapter 4 for more information.) To return to the OpenVMS Debugger, you enter "EXIT" at the "SDA>" prompt. Optionally, you may invoke SDA to perform a single command and then return immediately to the OpenVMS Debugger, as in the following example:

DBG> SDA SHOW SUMMARY

SDA may be reentered at any time, with or without the optional SDA command. Once SDA has been initialized, the SDA> prompt is output more quickly on subsequent occasions.

Note that there are some limitations on the use of SDA from within SDD:

- You cannot switch between processes, whether requested explicitly (SET PROCESS <name>) or implicitly (SHOW PROCESS <name>). The exception to this is that access to the system process is possible.

- You cannot switch between CPUs.

- SDA has no knowledge of the OpenVMS debugger’s Motif or Windows interfaces. Therefore, all SDA input and output occurs at the terminal or window where the OpenVMS debugger was originally invoked. Also, while using SDA, the OpenVMS debugger window is not refreshed; you must exit SDA to allow the OpenVMS debugger window to be refreshed.

- When you invoke SDA from SDD with an immediate command, and that command produces a full screen of output, SDA displays the message "Press RETURN for more." followed by the "SDA>" prompt before continuing. At this prompt, if you enter another SDA command, SDA does not automatically return to SDD upon completion. To do this, you must enter an EXIT command.

If the need arises to switch between processes or CPUs in the system dump, then you must invoke SDA separately using the DCL command ANALYZE/CRASH_DUMP.
9.8 Limitations of the System Dump Debugger

SDD provides a narrow window into the context of the system that was current at the time that the system crashed (stack, process, CPU, and so on). It does not provide full access to every part of the system as is provided by SDA. However, it does provide a view of the failed system using the semantics of the OpenVMS debugger—source correlation and display, call frame traversal, examination of variables by name, language constructs, and so on.

SDD therefore provides an additional approach to analyzing system dumps that is difficult to realize with SDA, often allowing quicker resolution of system crashes than is possible with SDA alone. When SDD cannot provide the needed data from the system dump, you should use SDA instead.

9.9 Access to Symbols in OpenVMS Executive Images

For a discussion and explanation of how the OpenVMS debugger accesses symbols in OpenVMS executive images, see Section 8.11.

9.10 Sample System Dump Debugging Session

This section provides a sample session that shows the use of some OpenVMS debugger commands as they apply to the system dump debugger. The examples in this section show how to work with a dump created as follows:

1. Follow the steps in Section 8.12, up to and including Example 8-9 (Breakpoint Display).
2. When the breakpoint at line 146 is reached, enter the OpenVMS debugger command to clear R27 and then continue:
   
   DBG> DEPOSIT R27=0
   DBG> GO

3. The system then crashes and a dump is written.
4. When the system reboots, copy the contents of SYS$SYSTEM:SYSDUMP.DMP to the build system with SDA:

   $ analyze/crash sys$system:sysdump.dmp

   OpenVMS (TM) Alpha system dump analyzer
   ...analyzing a selective memory dump...

   %SDA-W-NOTSAVED, global pages not saved in the dump file
   Dump taken on 1-JAN-1998 00:00:00.00
   INVECEPTN, Exception while above ASTDEL

   SDA> copy hstsys::sysdump.dmp
   SDA>

   To reproduce this sample session, you need access to the SYSTEM_DEBUG.DSF matching the SYSTEM_DEBUG.EXE file on your test system and to the source file C_TEST_ROUTINES.C, which is available in SYS$EXAMPLES.
Example 9–1 Invoking the System Dump Debugger

$ define dbg$decw$display " "
$ debug/keep

OpenVMS Alpha Debug64 Version V7.2-019

DBG>

Use the ANALYZE/CRASH_DUMP command to open the system dump. In this example, the logical name DBGHK$IMAGE_PATH is used for the image path, so the command qualifier /IMAGE_PATH is not being used. You may need to use it.

When you have opened the dump file, the DEBUG prompt is displayed. You should now do the following:

1. Set the language to be C, the language of the module that was active at the time of the system crash.

2. Set the source directory to the location of the source of the module. Use the debugger’s SET SOURCE command, which takes a search list as a parameter so you can make it point to many places.

Example 9–2 Accessing the System Dump

DBG> analyze/crash_dump sysdump.dmp
%SDA-W-NOTSAVED, global pages not saved in the dump file
%DEBUG-I-INIBRK, target system interrupted
%DEBUG-I-DYNIMGSET, setting image SYSTEM_DEBUG
%DEBUG-I-DYNMODSET, setting module C_TEST_ROUTINES
DBG> set language c
DBG> set source/latest sys$examples,sys$library
DBG>
Now that the debugger has access to the source, you can put the debugger into screen mode to see exactly where you are and the code surrounding it.

Example 9–3  Displaying the Source Code

DBG> Set Mode Screen; Set Step Nosource
  - SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
    135:   int i,k;
    136:   for(k=0;k<1000;k++)
    137:     {
    138:       test_c_code5(&i);
    139:     }
    140:   return;
    141: }  
    142: int test_c_code3(int subrtnCount)
    143: {
    144:   subrtnCount = subrtnCount - 1;
    145:   if (subrtnCount != 0)
->  146:     subrtnCount = test_c_code3(subrtnCount);
    147:   return subrtnCount;
    148: }
    149: int test_c_code2(int64 in64,int in32, int64 test, int64* pVar)
    150: {
    151:   xdt$fregsav[5] = in64;
    153:   if (xdt$fregsav[9] > 0)
    154:     *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
    155:   else
    156:     *pVar = (*pVar + xdt$fregsav[17]);
    157:   xdt$fregsav[7] = test_c_code3(10);
  - OUT -output-------------------------------------------------------------------
Now, you try a couple of other commands, EXAMINE and SHOW CALLS. The EXAMINE command allows you to look at all the C variables. Note that the C_TEST_ROUTINES module is compiled with the /NOOPTIMIZE switch which allows access to all variables. The SHOW CALLS command shows you the call sequence from the beginning of the stack. In this case, you started out in the image EXEC_INIT. (The debugger prefixes all images other than the main image with SHARE$ so it shows up as SHARE$EXEC_INIT.)

Example 9–4 Using the Examine and Show Calls Commands

```
DBG> Set Mode Screen; Set Step Nosource
-SRC: module C_TEST_ROUTINES -scroll-source------------------------------------
 135: int i,k;
 136: for(k=0;k<1000;k++)
 137:  {
 138:     test_c_code5(&i);
 139:  }
 140: return;
 141: }
 142: int test_c_code3(int subrtnCount)
 143: {
 144:     subrtnCount = subrtnCount - 1;
 145:     if (subrtnCount != 0)
 146:     -> subrtnCount = test_c_code3(subrtnCount);
 147:     return subrtnCount;
 148: }
 149: int test_c_code2(int64 in64,int in32, int test, int64* pVar)
 150: {
 151:     xdt$fregsav[5] = in64;
 153:     if (xdt$fregsav[9] > 0)
 154:         *pVar = (*pVar + xdt$fregsav[17])%xdt$fregsav[9];
 155:     else
 156:         *pVar = (*pVar + xdt$fregsav[17]);
 157:     xdt$fregsav[7] = test_c_code3(10);
-OUT -output-------------------------------------------------------------------
C_TEST_ROUTINES\test_c_code3\subrtnCount: 9
module name routine name line rel PC abs PC
*C_TEST_ROUTINES test_c_code3 146 00000000000000CC FFFFFFFF83002D6C
*C_TEST_ROUTINES test_c_code2 157 00000000000001A0 FFFFFFFF83002E40
*C_TEST_ROUTINES test_c_code 170 0000000000000260 FFFFFFFF83002F00
*XDELTA XDT$SYSDBG_INIT 9371 0000000000000058 FFFFFFFF83052238
*SYS$DOINIT INI$DOINIT 1488 0000000000000098 FFFFFFFF83052B8
SHARE$EXEC_INIT 0000000000018C74 FFFFFFFF83086C74
SHARE$EXEC_INIT 0000000000014BD0 FFFFFFFF83082BD0
-PROMPT -error-program-prompt--------------------------------------------------
DBG> e subrtnCount
DBG> show calls
DBG>
```
Part III
OpenVMS Watchpoint Utility

Part 3 describes the Watchpoint utility. It presents how to use the Watchpoint utility by doing the following:

• Loading the watchpoint driver
• Creating and deleting watchpoints
• Looking at watchpoint driver data
• Acquiring collected watchpoint data
• Looking at the protection attributes and access fault mechanism
• Looking at some watchpoint restrictions
This chapter describes the Watchpoint utility (WP), which enables you to monitor write access to user-specified locations. The chapter contains the following sections:

Section 10.1 presents an introduction of the Watchpoint utility.
Section 10.2 describes how to load the watchpoint driver.
Section 10.3 describes the creation and deletion of watchpoints and the constraints upon watchpoint locations.
Section 10.4 contains detailed descriptions of the watchpoint driver data structures, knowledge of which may be required to analyze collected watchpoint data.
Section 10.5 discusses acquiring collected watchpoint data.
Section 10.6 describes the watchpoint protection facility.
Section 10.7 describes its restrictions.

### 10.1 Introduction

A watchpoint is a data field to which write access is monitored. The field is from 1 to 8 bytes long and must be contained within a single page. Typically, watchpoints are in nonpaged pool. However, subject to certain constraints (see Section 10.3.1), they can be defined in other areas of system space. The Watchpoint facility can simultaneously monitor a large number (50 or more) watchpoints.

The utility is implemented in the WPDRIVER device driver and the utility program WP. This document concentrates on the device driver, which can be invoked directly or through the WP utility.

For information on the WP utility, see its help files, which can be displayed with the following DCL command:

```
$ HELP/LIBRARY=SYSSHELP:WP
```

Once the driver has been loaded, a suitably privileged user can designate a watchpoint in system space. Any write to a location designated as a watchpoint is trapped. Information is recorded about the write, including its time, the register contents, and the program counter (PC) and processor status longword (PSL) of the writing instruction. Optionally, one or both of the following user-specified actions can be taken:

- An XDELTA breakpoint\(^1\) or SCD breakpoint which occurs just after the write to the watchpoint

\(^1\) For simplicity, this chapter only mentions XDELTA. Any reference to XDELTA breakpoints also implies SCD breakpoints.
10.1 Introduction

- A fatal watchpoint bugcheck which occurs just after the write to the watchpoint.

You define a watchpoint by issuing QIO requests to the watchpoint driver; entering commands to the WP utility, which issues requests to the driver; or, from kernel mode code, invoking a routine within the watchpoint driver.

The WPDRIVER data structures store information about writes to a watchpoint. This information can be obtained either through QIO requests to the WPDRIVER, commands to the WP utility, XDELTA commands issued during a requested breakpoint, or SDA commands issued during the analysis of a requested crashdump.

10.2 Initializing the Watchpoint Utility

From a process with CMKRNL privilege, run the SYMSAN utility to load the watchpoint driver, SYS$WPDRIVER.EXE. Enter the following commands:

```plaintext
$ RUN SYS$SYSTEM:SYSMAN
SYSMAN> IO CONNECT WPA0:/NOADAPTER/DRIVER=SYS$WPDRIVER
SYSMAN> EXIT
```

SYSMAN creates system I/O data structures for the pseudo-device WPA0, loads WPDRIVER, and invokes its initialization routines. WPDRIVER initialization includes the following actions:

- Allocating nonpaged pool and physical memory for WPDRIVER data structures
- Appropriating the SCB vector specific to access violations
- Recording in system space the addresses of the WPDRIVER routines invoked by kernel mode code to create and delete watchpoints

Memory requirements for WPDRIVER and its data structures are:

- Device driver and UCB—approximately 3K bytes of nonpaged pool
- Trace table and a related array—36 bytes for each of system parameter WPTTE_SIZE trace table entries
- Watchpoint restore entries—system parameter WPRE_SIZE pages of physically contiguous memory
- Each watchpoint—176 bytes of nonpaged pool

It is advisable to load the watchpoint driver relatively soon after system initialization to ensure its allocation of physically contiguous memory. If the driver cannot allocate enough physically contiguous memory, it does not set WPA0: online. If the unit is offline, you will not be able to use the watchpoint utility.

10.3 Creating and Deleting Watchpoints

There are three different ways to create and delete watchpoints:

- An image can assign a channel to device WPA0: and then request the Queue I/O Request ($QIO) system service to create or delete a watchpoint.
- Code running in kernel mode can dispatch directly to routines within the WPDRIVER to create and delete watchpoints.
- You can enter commands to the WP utility.
The first two methods are described in detail in the sections that follow.

10.3.1 Using the $QIO Interface

An image first assigns a channel to the pseudo-device WPA0: and then issues a $QIO request on that channel. The process must have the privilege PHY_IO; otherwise, the $QIO request is rejected with the error SS_NOPRIV.

Table 10-1 shows the functions that the driver supports.

<table>
<thead>
<tr>
<th>Function</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_ACCESS</td>
<td>Creates a watchpoint</td>
</tr>
<tr>
<td>IO_DEACCESS</td>
<td>Deletes a watchpoint</td>
</tr>
<tr>
<td>IO_RDSTATS</td>
<td>Receives trace information on a watchpoint</td>
</tr>
</tbody>
</table>

The IO_ACCESS function requires the following device/function dependent arguments:

- P2—Length of the watchpoint. A number larger than 8 is reduced to 8.
- P3—Starting address of the watchpoint area.

The following are the constraints on the watchpoint area. It must be:

- Nonpageable system space.
- Write-accessible from kernel mode.
- Within one page. If it is not, the requested length is reduced to what will fit within the page containing the starting address.
- Within a page accessed only from kernel mode and by instructions that incur no pagefaults.
- Within a page whose protection is not altered while the watchpoint is in place.
- Outside of certain address ranges. These are the WPDRIVER code, its data structures, and the system page table.

Because of the current behavior of the driver, there is an additional requirement that there be no “unexpected” access violations referencing a page containing a watchpoint. See Section 10.7 for further details.

To specify that an XDELTA breakpoint or a fatal bugcheck occur if the watchpoint is written, use the following I/O function code modifiers:

- IO_M_CTRL to request an XDELTA breakpoint
- IO_M_ABORT to request a fatal bugcheck

For an XDELTA breakpoint to be taken, OpenVMS must have been booted specifying that XDELTA and/or the SCD be resident (bit 1 or bit 15 in the boot flags must be set). If both watchpoint options are requested, the XDELTA breakpoint is taken first. At exit from the breakpoint, the driver crashes the system.
A request to create a watchpoint can succeed completely, succeed partially, or fail. Table 10–2 shows the status codes that can be returned in the I/O status block.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$_NORMAL</td>
<td>Success.</td>
</tr>
<tr>
<td>SS$_BUFFEROVF</td>
<td>A watchpoint was established, but its length is less than was requested because the requested watchpoint would have straddled a page boundary.</td>
</tr>
<tr>
<td>SS$_EXQUOTA</td>
<td>The watchpoint could not be created because too many watchpoints already exist.</td>
</tr>
<tr>
<td>SS$_INSFMEM</td>
<td>The watchpoint could not be created because there was insufficient nonpaged pool to create data structures specific to this watchpoint.</td>
</tr>
<tr>
<td>SS$_IVADDR</td>
<td>The requested watchpoint resides in one of the areas in which the WPDRIVER is unable to create watchpoints.</td>
</tr>
<tr>
<td>SS$_WASSET</td>
<td>An existing watchpoint either coincides or overlaps with the requested watchpoint.</td>
</tr>
</tbody>
</table>

The following example MACRO program assigns a channel to the WPA0 device and creates a watchpoint of 4 bytes, at starting address 80001068. The program requests neither an XDELTA breakpoint nor a system crash for that watchpoint.

```
$IODEF
  .PSECT RWDATA,NOEXE,RD,WRT,LONG
  WP_IOSB: .BLKL 2 ; I/O status block.
  WP_ADDR: .LONG ^X80001068 ; Address of watchpoint to create.
  WP_NAM: .ASCID /WPA0:/ ; Device to which to assign channel.
  WP_CHAN: .BLKW 1 ; Channel number.
  .PSECT PROG,EXE,NOWRT

START: .CALL_ENTRY
  $ASSIGN_S DEVNAM=WP_NAM,CHAN=WP_CHAN
  BLBC R0,RETURN

  $QION_S CHAN=WP_CHAN, -
    FUNC=#I0$_ACCESS, -
    IOSB=WP_IOSB, -
    P2=##4, -
    P3=WP_ADDR
  BLBC R0,RETURN
  MOVL WP_IOSB,R0 ; Move status to R0.
RETURN: RET ; Return to caller.
.END START
```

A watchpoint remains in effect until it is explicitly deleted. (Note, however, that watchpoint definitions do not persist across system reboots.) To delete an existing watchpoint, issue an IO$_DEACCESS QIO request.

The IO$_DEACCESS function requires the following device/function dependent argument: P3 - Starting address of the watchpoint to be deleted.
Table 10–3 shows the status values that are returned in the I/O status block.

<table>
<thead>
<tr>
<th>Status Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS$NORMAL</td>
<td>Success.</td>
</tr>
<tr>
<td>SS$IVADDR</td>
<td>The specified watchpoint does not exist.</td>
</tr>
</tbody>
</table>

Section 10.5 describes the use of the IO$RDSTATS QIO request.

### 10.3.2 Invoking WPDRIVER Entry Points from System Routines

When the WPDRIVER is loaded, it initializes two locations in system space with the addresses of routines within the driver. These locations, WP$CREATE_WATCHPOINT and WP$DELETE_WATCHPOINT, enable dispatch to create and delete watchpoint routines within the loaded driver. Input arguments for both routines are passed in registers.

Code running in kernel mode can execute the following instructions:

```
JSB @G^WP$CREATE_WATCHPOINT ; create a watchpoint
```

and

```
JSB @G^WP$DELETE_WATCHPOINT ; delete a watchpoint
```

Both these routines save IPL at entry and set it to the fork IPL of the WPDRIVER, IPL 11. Thus, they should not be invoked by code threads running above IPL 11. At exit, the routines restore the entry IPL.

These two locations contain an RSB instruction prior to the loading of the driver. As a result, if a system routine tries to create or delete a watchpoint before the WPDRIVER is loaded, control immediately returns.

**WP$CREATE_WATCHPOINT** has the following register arguments:

- **R0**—User-specified watchpoint options
  - Bit 1 equal to 1 specifies that a fatal OPERCRASH bugcheck should occur after a write to the watchpoint area.
  - Bit 2 equal to 1 specifies that an XDELTA breakpoint should occur after a write to the watchpoint area.
- **R1**—Length of the watchpoint area
- **R2**—Starting address of the watchpoint area

Status is returned in R0. The status values and their interpretations are identical to those for the QIO interface to create a watchpoint. The only difference is that the SS$NOPRIV status cannot be returned with this interface.

**WP$DELETE_WATCHPOINT** has the following register argument:

- **R2**—Starting address of the watchpoint area

Status is returned in R0. The status values and their interpretations are identical to those for the QIO interface.
10.4 Data Structures

The WPDRIVER uses three different kinds of data structures:

- One watchpoint restore entry (WPRE) for each page of system space in which one or more active watchpoints are located
- One watchpoint control block (WPCB) for each active watchpoint
- Trace table entries (WPTTEs) in a circular trace buffer which maintains a history of watchpoint writes

These data structures are described in detail and illustrated in the sections that follow.

10.4.1 Watchpoint Restore Entry (WPRE)

There is one WPRE for each system page that contains a watchpoint. That is, if nine watchpoints are defined which are in four different system pages, four WPREs are required to describe those pages. When WPDRIVER is loaded, its initialization routine allocates physically contiguous memory for the maximum number of WPREs. The number of pages to be allocated is specified by system parameter WPRE_SIZE.

The WPDRIVER allocates WPREs starting at the beginning of the table and maintains a tightly packed list. That is, when a WPRE in the middle of those in use is “deallocated,” its current contents are replaced with the contents of the last WPRE in use. The number in use at any given time is in the driver variable WP$L_WP_COUNT. The system global EXE$GA_WP_WPRE points to the beginning of the WPRE table.

The WPRE for a page contains information useful for:

- Determining whether a given access violation refers to an address in the page associated with this WPRE
- Restoring the original SPTE value for the associated page
- Reestablishing the modified SPTE value when watchpoints are reenabled
- Invalidating the translation buffer when the SPTE is modified
- Locating the data structures associated with individual watchpoints defined in this system page

10.4.2 Watchpoint Control Blocks (WPCB)

The WPCBs associated with a given system page are singly-linked to a list header in the associated WPRE. A WPCB is allocated from a nonpaged pool when a watchpoint is created. A WPCB contains static information about the watchpoint such as the following:

- Its starting address and length
- Original contents of the watchpoint at the time it was established
- User-specified options for this watchpoint

In addition, the WPCB contains dynamic data associated with the most recent write reference to the watchpoint. This data includes the following:

- Number of times that the watchpoint has been written.
- Address of the first byte within the watchpoint that was modified at the last write reference.
• PC-PSL pair that made the last write reference.
• System time at the last write reference.
• Contents of the general registers at the time of the last write reference.
• A copy of up to 15 bytes of instruction stream data beginning at the program counter (PC) of the instruction that made the last write reference. The amount of instruction stream data that is copied here is the lesser of 15 bytes and the remaining bytes on the page containing the PC.
• Contents of the watchpoint before the last write reference.
• Contents of the watchpoint after the last write reference. This value is presumably the current contents of the watchpoint.
• A pointer to an entry in the global circular trace buffer where all recent references to watchpoints are traced.

10.4.3 Trace Table Entries (WPTTEs)
Whenever a watchpoint is written, all the relevant data is recorded in the WPCB associated with the watchpoint. In addition, to maintain a history, the WPDRIVER copies a subset of the data to the oldest WPTTE in the circular trace buffer. Thus, the circular trace buffer contains a history of the last N references to watchpoints. The driver allocates nonpaged pool to accommodate the number of trace table entries specified by the system parameter WPTTE_SIZE. The WPTTEs for all watchpoints are together in the table, but the ones for a particular watchpoint are chained together.

The subset of data in a WPTTE includes the following:
• Starting address of the watchpoint
• Relative offset of the first byte modified on this reference
• Opcode of the instruction that modified the watchpoint
• A relative backpointer to the previous WPTTE of this watchpoint
• PC-PSL of the write reference
• System time of the write reference
• Contents of the watchpoint before this reference

10.5 Analyzing Watchpoint Results
Analyzing watchpoint results is a function of the mode in which the WPDRIVER is used. For example, if you have only one watchpoint and have specified that an XDELTA breakpoint and/or a bugcheck occur on a write to the watchpoint, then when the reference occurs, simply find the program counter (PC) that caused the reference.

This PC (actually the PC of the next instruction) and its processor status longword (PSL) are on the stack at the time of the breakpoint and/or bugcheck. The layout that follows is the stack as it appears within an XDELTA breakpoint. Examined from a crash dump, the stack is similar but does not contain the return address from the JSB to INI$BRK.
Furthermore, R0 contains the address of the WPCB associated with that watchpoint. You can examine the WPCB to determine the original contents of the watchpoint area and the registers at the time of the write.

Definitions for the watchpoint data structures are in SYS$LIBRARY:LIB.MLB. Build an object module with its symbol definitions by entering the following DCL commands:

```
$ MACRO/OBJ=SYS$LOGIN:WPDEFS SYS$INPUT: + SYS$LIBRARY:LIB/LIB
$WPCBDEF GLOBAL !n.b. GLOBAL must be capitalized
$WPREDEF GLOBAL
$WPTTEDEF GLOBAL .END
CTRL/Z
```

Then, within SDA, you can format watchpoint data structures. For example, enter the following SDA commands:

```
SDA> READ SYS$LOGIN:WPDEFS.OBJ
SDA> FORMAT @R0 /TYPE=WPCB !type definition is required
SDA> DEF WPTTE = @R0 + WPCB$L_TTE
SDA> FORMAT WPTTE /TYPE=WPTTE
```

An alternative to crashing the system or using XDELTA to get watchpoint information is the QIO function IO$_RDSTAT. This function returns watchpoint control block contents and trace table entries for a particular watchpoint.

It requires the following device/function dependent arguments:

- P1—Address of buffer to receive watchpoint data.
- P2—Length of the buffer. The minimum size buffer of 188 bytes is only large enough for WPCB contents.
- P3—Watchpoint address.

The data returned in the buffer has the format shown in Figure 10–1.
10.6 Watchpoint Protection Overview

The overall design of the watchpoint facility uses protection attributes on system pages and the access violation fault mechanism. To establish a watchpoint within a page of system space, the WPDRIVER changes the protection of the page to disallow writes. The WPDRIVER modifies the access violation vector to point to its own routine, WP$ACCVIO.

Any subsequent write to this page causes an access violation and dispatch to WP$ACCVIO. Thus, the WPDRIVER gains control on all write references to watchpoints and can monitor such accesses.

When WP$ACCVIO is entered, it raises IPL to 31 to block all other threads of execution. It first must determine whether the faulting address (whose reference caused the access violation) is within a page containing a watchpoint. However, any major amount of CPU processing at this point might access an area in system space whose protection has been altered to establish watchpoints. As a result, such processing might cause a reentry into WP$ACCVIO. To avoid recursive reentry, WP$ACCVIO first restores all SPTEs that it had modified to their values prior to the establishment of any watchpoints. From this point until this set of SPTEs are remodified, no watchpoints are in effect. Now WP$ACCVIO can determine whether the reference was to a page containing a watchpoint.

To determine whether the reference is to a watchpoint page, WP$ACCVIO compares the faulting address to addresses of pages whose protection has been altered by WPDRIVER. If the faulting address is not in one of these pages, then WP$ACCVIO passes the access violation to the usual OpenVMS service routine, EXE$ACVIOLAT. If the faulting address is within a page containing a watchpoint, more extensive processing is required.

As a temporary measure, WP$ACCVIO first records all data related to the reference in its UCB. It cannot immediately associate the access violation with a particular watchpoint. This ambiguity arises from imprecision in the faulting virtual address recorded at the access violation. The CPU need merely place on the stack “some virtual address in the faulting page.”
As a result, when a reference to a page with a watchpoint results in an access violation, the watchpoint driver first merely captures the data in its UCB. The data captured at this point includes the following:

- PC and PSL of the faulting instruction
- Current system time
- Values of all the general registers from R0 through SP
- A copy of up to 15 bytes of the instruction stream, beginning at the PC previously captured

If the reference later turns out not to be one to a watchpoint, the captured data is discarded. If the reference is to a watchpoint, the data is copied to the WPCB and circular trace buffer.

The watchpoint driver distinguishes between these two possibilities by reexecuting the faulting instruction under a controlled set of circumstances.

Once the instruction has reexecuted, WP$TBIT can determine whether watchpoint data has been modified by comparing the current contents of all watchpoints within the page of interest to the contents that they had prior to this reference. Because the driver has run at IPL 31 since the write access that caused an access violation, any change in the contents is attributable to the reexecuted instruction. If the contents of a watchpoint are different, WP$TBIT copies the data temporarily saved in its UCB to the WPCB associated with this watchpoint and records a subset of this data in a WPTTE.

The driver can cause either or both an XDELTA breakpoint or a bugcheck, depending on what action was requested with the watchpoint definition. If an XDELTA breakpoint was requested, the driver invokes XDELTA. After the user proceeds from the XDELTA breakpoint, if a bugcheck was not requested, the driver restores the SPTEs of pages containing watchpoints, the saved registers and IPL, and REIs to dismiss the exception.

### 10.7 Restrictions

The WPDRIVER can monitor only those write references to system space addresses that arise in a CPU. I/O devices can write to memory and thereby modify watchpoints without the WPDRIVER’s becoming aware of the write.

Because a write access to a watchpoint is determined by comparing the contents of the watchpoint before and after the write, a write of data identical to the original contents is undetectable.

Because the WPDRIVER modifies SPTEs, a device page that directly interprets tables may experience access violations when it attempts to write into a memory page whose protection has been modified to monitor watchpoints. In other words, a page containing a watchpoint should not also contain a buffer for such a controller.

When you create a watchpoint, you should ensure that the system is quiet with respect to activity affecting the watchpoint area. Otherwise, an inconsistent copy of the original contents of the watchpoint area may be saved. WPDRIVER raises IPL to 11 to copy the watchpoint area’s original contents. This means that if the area is modified from a thread of execution running as the result of an interrupt above 11, WPDRIVER can copy inconsistent contents. An inconsistent copy of the original contents may result in spuriously detected writes and missed writes.
If the page containing the watchpoint area is written by an instruction that incurs a page fault, the system can crash with a fatal PGFIPLHI bugcheck. As described in the previous section, after detecting an attempt to write to a page with a watchpoint, the WPDRIVER re-executes the writing instruction at IPL 31. Page faults at IPL 31 are not allowed.

If an outer access mode reference to a watchpointed page causes an access violation, the system will likely crash. When an access violation occurs on a page with a watchpoint, the current driver does not probe the intended access and faulting mode against the page's original protection code. Instead, it assumes that any access violation to that page represents a kernel mode instruction that can be reexecuted at IPL 31. The driver’s subsequent attempt to REI, restoring a program status longword (PSL) with an outer mode and IPL 31, causes a reserved operand fault and, generally, a fatal INVECEPTN bugcheck.

You must be knowledgeable about the accesses to the page with the watchpoint and careful in using the driver. You should test the watchpoint creation on a standalone system. You should leave the watchpoint in effect long enough to have some confidence that pagefaults in instructions accessing that page are unlikely.

An attempt to CONNECT a WPA unit other than zero results in a fatal WPDRVRRERR bugcheck.

The WPDRIVER is suitable for use only on a single CPU system. That is, it should not be used on a symmetric multiprocessing system. There are no plans to remove this restriction in the near future.
Index

A

Access rights block, 2–19
Access violations, 2–25, 2–26
ACME.EXE file, 4–35
ACPs (ancillary control processes), 4–82
Addition operator (+), 2–16
Addresses, examining, 4–18
Address operator (@), 2–15
Address operator (^B), 2–15
Address operator (^L), 2–15
Address operator (^P), 2–15
Address operator (^Q), 2–15
Address operator (^V), 2–15
Address operator (^W), 2–15
Address space number (ASN), 2–18
ANALYZE command
 /CRASH_DUMP qualifier, 2–10, 3–1, 3–3
 /OVERRIDE qualifier, 3–4
 /RELEASE qualifier, 3–5
 /SYMBOL qualifier, 3–6
 /SYSTEM qualifier, 2–2, 3–1, 3–7
Analyzing watchpoint results, 10–7
Ancillary control process
 See ACPs
AND operator (&), 2–16
AOBs (ACP queue blocks), 4–83
ARB symbol, 2–19
Arithmetic operators, 2–15
Arithmetic shifting operator (@), 2–16
ASBs (asynchronous save blocks), 4–56
ASN register displaying, 4–75
ASTEN register, displaying, 4–75
ASTSR register, displaying, 4–75
AST (asynchronous system traps), 2–18
AST symbols, 2–18
Asynchronous save blocks
 See ASBs
Asynchronous system traps
 See ASTs
At sign (@)
 as execute command, 4–3
 as shifting operator, 2–16
ATTACH command, 4–4
 /PARENT qualifier, 4–4

B

Backup utility (BACKUP), copying system dump file, 2–8
BDBs (buffer descriptor blocks), 4–56
BDBSUM (BDB summary page), 4–56
Binary operators, 2–16
BLBs (buffer lock blocks), 4–56
BLBSUM (BLB summary page), 4–56
Buffer descriptor blocks
 See BDBs
Buffer lock blocks
 See BLBs
Bugcheck
 code, 2–22
 fatal conditions, 2–22 to 2–33
 halt/restart, 2–10
 handling routines
 global symbols, 4–35
 reasons for taking, 4–78

C

Call frames
 displaying in SDA, 4–64
 following a chain, 4–64
Cancel I/O routine, 4–82
Catenate operator (.), 2–16
CCBs (channel control blocks), displaying in SDA, 4–56
CDDBs (class driver data blocks), 4–83
CDRPs (class driver request packets), 4–72, 4–183
CDTs (connection descriptor tables), 4–72, 4–183
Channel control blocks
 See CCBs
 /CHANNEL qualifier, 4–159
Channel request blocks
 See CRBs
Class driver data blocks
 See CDDBs
Class driver request packets
 See CDRPs
CLUBs (cluster blocks), 4–67
CLUFCBs (cluster failover control blocks), 4–67
Cluster blocks
  See CLUBs
Cluster failover control blocks
  See CLUFCBs
Cluster quorum disk control blocks
  See CLUDCBs
Cluster system blocks
  See CSBs
Cluster system identification numbers
  See CSIDs
CNX$DEBUG.EXE file, 4–35
Compressed data section, 4–29
Condition-handling routines, global symbols, 4–35
Condition values, evaluating, 4–15
Connection descriptor tables
  See CDTs
Connection manager, displaying SDA information, 4–66
Connections, displaying SDA information about, 4–72, 4–146, 4–183
Contents of stored machine check frames
  displaying in SDA, 4–121
Context
  SDA CPU, 2–13
  SDA process, 2–12
Control blocks, formatting, 4–23
Control region, 2–18
  examining, 4–19
Control region operator (H), 2–16
COPY command, 2–7, 2–8, 4–5
  /COMPRESS qualifier, 4–5
  /DECOMPRESS qualifier, 4–5
  /CPU= qualifier, 4–77
CPU context
  changing, 4–54
    using SET CPU command, 4–45
    using SHOW CPU command, 4–74
    using SHOW CRASH command, 4–77
    using SHOW PROCESS command, 4–150
  displaying, 4–74
CPUDB symbol, 2–19
CPU ID
  See CPU identification number
CPU identification number, 4–74
Crash dumps
  file headers, 4–105
    headers, 4–105
    incomplete, 2–10
    short, 2–10
CRBs (channel request blocks), 4–82
CREATE command, 2–7
Creating and deleting watchpoints, 10–2
CSBs (cluster system blocks), 4–67, 4–72

Index–2
CSIDs (cluster system identification numbers), 4-66, 4-177
Current stack pointer, 2-19

D

Data structures
  formatting, 4-23
  global symbols, 2-18
  stepping through a linked list, 4-41
DCLDEF.STB file, 2-18
DCL interpreter, global symbols, 2-18
DBBs (device data blocks), 4-82
DDIF$RMS_EXTENSION.EXE file, 4-35
DDTs (driver dispatch tables), 4-82
Debugging system image, 8-2
DECDTMDSTB.STB file, 2-18
Decimal value of an expression, 4-15
DECnet, global symbols, 2-18
DEFINE command, 4-7, 4-9
  /IF_STATE qualifier, 4-10
  /KEY qualifier, 4-10
  /LOCK_STATE qualifier, 4-10
  /PD qualifier, 4-7
  /SET_STATE qualifier, 4-10
DEFINE command/TERMINATE qualifier, 4-10
Delta/XDelta Debugger (DELTA/XDELTA), 1-2
Device data blocks
  See DDBs
Device driver routines, address, 4-82
Devices, displaying SDA information, 4-81
Division operator (/), 2-16
DOSD (dump off system disk), 1-3, 5-2
DPTs (driver prologue tables), 4-82
Driver dispatch tables
  See DDTs
Driver prologue tables
  See DPTs
DUMPBUG system parameter, 2-3, 2-34
DUMP command, 4-12
  /COUNT = [ALL | records] qualifier, 4-12
  /DECIMAL qualifier, 4-12
  /FORWARD qualifier, 4-12
  /HEXADECIMAL qualifier, 4-12
  /INDEX_ARRAY [= {LONGWORD | QUADWORD}], 4-12
  /INITIAL_POSITION qualifier, 4-12
  /LONGWORD qualifier, 4-12
  /PHYSICAL qualifier, 4-13
  /QUADWORD qualifier, 4-13
  /RECORD_SIZE= size qualifier, 4-13
  /REVERSE qualifier, 4-13
Dump file
  analyzing, 3-1
  copying, 4-5
  displaying a summary of, 5-9
  displaying machine check information, 5-16
  displaying memory with CLUE MEMORY, 5-17
  displaying process information, 5-25
  displaying the current stack, 5-30
  displaying virtual I/O cache, 5-34
  displaying XQP information, 5-37
  extracting errorlog buffers, 5-12
  purging files using CLUE CLEANUP, 5-7
  saving automatically, 2-8, 5-1
  saving output, 5-14
  using CLUE CONFIG, 5-8
DUMPSTYLE system parameter, 2-5
DUMP subset, 2-5

E

ERRORLOG.STB file, 4-35
ERRORLOGBUFFERS system parameter, 2-7
Error logging
  global symbols, 4-35
  routines, 4-35
Error log messages, 5-12
ESP symbol, 2-18
EVALUATE command, 4-15
  /CONDITION_VALUE qualifier, 4-15
  /PS qualifier, 4-15
  /PTE qualifier, 4-15
  /SYMBOLS qualifier, 4-15
  /TIME qualifier, 4-15
EXAMINE command, 4-18
  /ALL qualifier, 4-18
  /CONDITION_VALUE qualifier, 4-18
  /INSTRUCTION qualifier, 4-18
  /NOPD qualifier, 4-18
  /NOSUPPRESS qualifier, 4-19
  /P0 qualifier, 4-19
  /P1 qualifier, 4-19
  /PD qualifier, 4-19
  /PHYSICAL qualifier, 4-19
  /PS qualifier, 4-19
  /PTE qualifier, 4-19
  /SYSTEM qualifier, 4-19
  /TIME qualifier, 4-20
EXCEPTION.STB file global symbols, 4-35
Exception-handling routines, global symbols, 4-35
Executive images
  contents, 4-35, 4-88
  global symbols, 4-33
Executive stack pointer, 2-18
EXEC_INIT.STB file, 4-35
EXIT command, 4-22
Exiting from SDA, 4-22
Expressions, 2-14
  evaluating, 4-15
Extended attribute blocks
  See XABs

Index–3
F
F11BXQP.STB file, 4–35
FABs (file access blocks), 4–56
Fatal exceptions, 2–22
FATALEXCPT bugcheck, 2–23
FC$GLOBALS.STB file, 4–35
FCBs (file control blocks), 4–56
FEN symbol, 2–18
File access blocks
See FABs
File control blocks
See FCBs
File statistics blocks
See FSB
File systems global symbols, 4–35
File work areas
See FWAs
Floating point
control register, 2–18
enable, 2–18
registers, 2–18
FORMAT command, 4–23
/PHYSICAL qualifier, 4–23
/TYPE qualifier, 4–23
FPCR register displaying, 4–75
FPCR symbol, 2–18
FP symbol, 2–18
Frame pointers, 2–18
FSB (file statistics block), 4–56
Full and selective dumps, 2–4
FWAs (file work areas), 4–56

G
GBDs (global buffer descriptors), summary page, 4–56
GBHs (global buffer headers), 4–56
GBHSH (global buffer hash table), 4–56
GBSBs (global buffer synchronization blocks), 4–56
Global buffer descriptors
See GBDs
Global buffer hash table
See GBHSH
Global buffer headers
See GBHs
Global buffer synchronization blocks
See GBSBs
Global page tables, displaying, 4–125
G operator, 2–15
G symbol, 2–18

H
Headers, crash dump, 4–105
HELP command, 4–26
recording output, 4–50
Hexadecimal value of an expression, 4–15
H operator, 2–16
H symbol, 2–18

I
I/O databases
displaying SDA information, 4–81
global symbols, 2–18
I/O request packets
See IRPs
IDBs (interrupt dispatch blocks), 4–82
IDXs (index descriptors), 4–56
IFABs (internal file access blocks), 4–56
IFIs (internal file identifiers), 4–56
Image activator, global symbols, 2–18, 4–36
IMAGE_MANAGEMENT.STB file, global symbols, 4–36
IMGDEF.STB file, 2–18
Implementing the Watchpoint utility, WPDRIVER, 10–1
Index descriptors
See IDXs
/IINDEX qualifier, 4–53
Initialization code global symbols, 4–35
Initializing Watchpoint utility, 10–2
Interlocked queues, validating, 4–213
Internal file access blocks
See IFABs
Internal file identifiers
See IFIs
Interrupt dispatch blocks
See IDBs
INVEXCEPTN bugcheck, 2–23
IODEF.STB file, 2–18
I operator, 2–16
I0_ROUTINES.STB file global symbols, 4–36
IPL register displaying, 4–75
IPL symbol, 2–19
IRABs (internal record access blocks), 4–56
IRPs (I/O request packets), 4–82
I symbol, 2–19

J
JFBs (journaling file blocks), 4–56
JBs (job information blocks), 4–156
JB symbol, 2–19
Job information block
See JIB
Journaling file blocks
See J FBs

K
Kernel stacks
displaying contents, 4–196
pointer, 2–19
Kernel threads block, 2–19
Kernel Threads Block
KTB, 4–156
Key-less-than block
See KLTB
Keys (keyboard), defining for SDA, 4–9
KLTB (key-less-than block), 4–56
KSP symbol, 2–19
KTB
kernel threads block, 4–156
KTB symbol, 2–19

L
LAT$RATING.EXE file, 4–36
LCK$DEBUG.EXE file, 4–36
Linker map, use in crash dump analysis, 2–22
Linking two 32-bit values ("."), 2–16
LKB (lock block), 4–118
LMF$GROUP_TABLE.EXE file, 4–36
Location in memory
examining, 4–18
SDA default, 4–18
translating to instruction, 4–18
LOCKING.STB file, 4–36
Lock management routines, global symbols, 4–36
Lock manager, displaying SDA information, 4–116
Locks, displaying SDA information, 4–177
Logical operators, 2–15, 2–16
AND operator (\&), 2–16
NOT operator (#), 2–15
OR operator ( | ), 2–16
XOR operator ( \ ), 2–16
LOGICAL_NAMES.STB file global symbols, 4–36
Lookaside lists displaying contents, 4–139

M
Machine check frames displaying in SDA, 4–121
MAP command, 4–28
MCES register displaying, 4–75
Mechanism arrays, 2–23
Memory
examining, 4–18
formatting, 4–23
locations
decoding, 4–20
examining, 4–20
region
examining, 4–21
MESSAGE_ROUTINES.STB file global symbols, 4–36
MODIFY DUMP command, 4–31
/BLOCK=n qualifier, 4–31
/BYTE command, 4–31
/CONFIRM=n qualifier, 4–31
/LONGWORD qualifier, 4–31
/NEXT qualifier, 4–31
/OFFSET=n qualifier, 4–31
/QUADWORD qualifier, 4–31
/WORD qualifier, 4–31
MSCP.EXE file, 4–36
MULTIPATH.STB file, 4–36
Multiplication operator (*), 2–16
Multiprocessing, global symbols, 4–38
Multiprocessors
analyzing crash dumps, 2–12
displaying synchronization structures, 4–190

N
NAMs (name blocks), 4–57
Negative operator ( – ), 2–15
NET$CSMACD.EXE file, 4–36
NET$FDI.EXE file, 4–36
NETDEF.STB file, 2–18
Nonpaged dynamic storage pool, displaying
contents, 4–139
NOT operator (#), 2–15
NT_EXTENSION.EXE file, 4–36
NWA (network work area), 4–57

O
Object rights block, 2–19
OpenVMS Cluster environments
displaying SDA information, 4–66
OpenVMS Cluster environments, displaying SDA information, 4–66
OpenVMS Galaxy data structures, symbols, 2–18
OpenVMS RMS
See RMS
Operators (mathematical)
precedence of, 2–15, 2–16
ORB symbol, 2–19
OR operator ( | ), 2–16

P
P0 region, examining, 4–19
P1 region, examining, 4–19
Paged dynamic storage pool displaying contents, 4–139
Page faults, illegal, 2–33
Page files
See also SYS$SYSTEM:PAGEFILE.SYS file
Page table base register, 2–19
Page table entries
  See PTEs
Page tables, displaying, 4–125, 4–152
Parentheses ( ), as precedence operators, 2–16
PB (path block), 4–82
PCBB register displaying, 2–19, 4–75
PCBB symbol, 2–19
PCBs (process control blocks), 2–19
  displaying, 4–153
  hardware, 4–157
  specifying the address of, 4–53, 4–150
PCB symbol, 2–19
PCC (process cycle counter), 2–19
PCC symbol, 2–19
PCs (program counters), 2–19
  in a crash dump, 2–22
PC symbol, 2–19
PDTs (port descriptor tables), 4–146
PFN (page frame number)
  See also PFN database
PFN database, displaying, 4–128, 4–134
PGFPLHLI bugcheck, 2–33
PHDs (process headers), 2–19, 4–154
PHD symbol, 2–19
Physical address operator ( ^P ), 2–15
PID numbers, 4–152
PIO, Use process-permanent I/O data structures, 4–57
Port drivers, displaying SDA information, 4–66
Positive operator (+), 2–15
PRBR register displaying, 4–75
PRBR symbol, 2–19
Precedence operators, 2–16
Privileges
  to analyze a crash dump, 3–1
  to analyze a running system, 2–12, 3–1
Process contexts, changing, 4–46, 4–53, 4–77, 4–150
Process control blocks
  See PCBs and System PCBs
  See system PCBs
Process control region, 2–18
  operator (H), 2–16
Processes
  displaying
    SDA information, 4–150, 4–200
    examining hung, 2–12
    image, 4–200
    listening, 4–67
    lock [brief], 4–152
    scheduling state, 4–156, 4–201
    spawning a subprocess, 4–208
    system, 4–53
Process indexes, 4–152
Process names, 4–150
Processor base registers, 2–19
Processor context, changing, 4–45, 4–54, 4–74, 4–77, 4–150
Processor status
  See PS
Process section tables
  See PSTs
PROCESS_MANAGEMENT.STB file global symbols, 4–36
Program regions, examining, 4–19
PS (process status)
  evaluating, 4–15
  examining, 4–19
PS symbol, 2–19
PSTs (process section tables) displaying, 4–154
PTBR register displaying, 4–75
PTBR symbol, 2–19
PTEs (page table entries)
  evaluating, 4–15
  examining, 4–19
Q
QSRV$GLOBALS.STB file, 4–36
Queues
  stepping through, 4–41
  validating, 4–213
Quorum, 4–66
R
RABs (record access blocks), 4–57
Radixes, default, 2–15
Radix operators, 2–15
RDTs (response descriptor tables), 4–183
READ command, 4–34
  /EXECUTIVE qualifier, 4–33
  /FORCE qualifier, 4–33
  /IMAGE qualifier, 4–34
  /LOG qualifier, 4–34
  /NOLOG qualifier, 4–34
  /RELOCATE qualifier, 4–34
  /SYMVA qualifier, 4–34
  SYS$DISK, 4–35
Record access blocks
  See RABs
Record lock blocks
  See RLBs
Recovery unit blocks
  See RUBs
Recovery unit file blocks
  See RUFBs
Recovery unit stream blocks
  See RUSBs
Index

Recovery unit system services, global symbols, 4–36
RECOVERY_UNIT_SERVICES.STB file, global symbols, 4–36
Registers
  displaying, 4–75, 4–154
  integer, 2–19
REPEAT command, 4–41
  /UNTIL=condition qualifier, 4–41
Report system event, global symbols, 4–36
REQSYSDEF.STB file, 2–18
Resident images, 4–151, 4–161
  /RESIDENT qualifier installing an image, 4–29
Resource blocks
  See RSBs
Resources, displaying SDA information, 4–175
Response descriptor tables
  See RDTs
Response ID
  See RSPID
RLBs (record lock blocks), 4–57
RMS
  data structures shown by SDA, 4–56
  displaying data structures, 4–155, 4–182
  global symbols, 2–18, 4–36
RMS.STB file, 4–36
RMSDEF.STB file, 2–18
RSBs (resource blocks), 4–118, 4–176
RSPID (response ID), displaying SDA information, 4–183
RUBs (recovery unit blocks), 4–57
RUFBs (recovery unit file blocks), 4–57
RUSBs (recovery unit stream blocks), 4–57

S
S0 region, examining, 4–19
SAVEDUMP system parameter, 2–7, 2–34
SBs (system blocks), 4–67, 4–82
SCBB register, displaying, 4–75
SCBB symbol, 2–19
SCC (system cycle counter), 2–19
SCC symbol, 2–19
SCD
  See System Code Debugger
Schedulers, global symbols, 4–36
SCS (System Communications Services)
  displaying SDA information, 4–66, 4–67, 4–72, 4–146, 4–183
  global symbols, 2–18
SCSDEF.STB file, 2–18
SDA$ADD_SYMBOL callable routine, 7–9
SDA$ALLOCATE callable routine, 7–10
SDA$DBG_IMAGE_INFO callable routine, 7–11
SDA$DEALLOCATE callable routines, 7–12
SDA$DISPLAY_HELP callable routine, 7–13
SDA$ENSURE callable routine, 7–15
SDA$FORMAT callable routine, 7–16
SDA$FORMAT HEADING callable routine, 7–18
SDA$GETMEM callable routine, 7–19
SDA$HEADER callable routine, 7–20
SDA$IMAGE OFFSET callable routine, 7–22
SDA$SAVE current CPU callable routine, 7–24
SDA$SAVE current PCB callable routine, 7–25
SDA$SAVE current PCB callable routine, 7–26
SDA$SAVE current PCB callable routine, 7–28
SDA$SAVE current PCB callable routine, 7–29
SDA$SAVE current process callable routine, 7–31
SDA$SAVE current process callable routine, 7–32
SDA$SAVE current process callable routine, 7–34
SDA$SAVE current process callable routine, 7–35
SDA$SAVE current process callable routine, 7–37
SDA$SAVE current process callable routine, 7–38
SDA$SAVE current process callable routine, 7–40
SDA$SAVE current process callable routine, 7–42
SDA$SAVE current process callable routine, 7–44
SDA$SAVE current process callable routine, 7–46
SDA$SAVE current process callable routine, 7–47
SDA$SAVE current process callable routine, 7–48
SDA$SAVE current process callable routine, 7–50
SDA$SAVE current process callable routine, 7–51
SDA$SAVE current process callable routine, 7–52
SDA$SAVE current process callable routine, 7–54
SDA$SAVE current process callable routine, 7–55
SDA$SAVE current process callable routine, 7–56
SDA$SAVE current process callable routine, 7–58
SDA$SAVE current process callable routine, 7–59
SDA, invoking by default, 2–8
SDA capabilities, 2–1
SDA CLUE, dump off system disk, 5–2
SDA CLUE commands
  archiving dump file information, 5–1
  collecting dump file information, 5–1
SDA command format, 2–14
SDA current CPU, 2–13, 4–45, 4–54, 4–74, 4–77, 4–150, 4–197
SDA current process, 2–13, 4–46, 4–53, 4–77, 4–150, 4–197
SDA symbol table
  building, 2–11
  expanding, 2–11
SDD
  See System Dump Debugger
SEARCH command, 4–43
/LENG qualifier, 4–43
/MASK qualifier, 4–43
/PHYLICAL qualifier, 4–43
/STEPS qualifier, 4–43
Section type, 4–151, 4–161
SECURITY.STB file global symbols, 4–36
Self-relative queue, validating, 4–213
SET CPU command, 2–13, 4–45
  analyzing a running system, 2–12
SET ERASE SCREEN command, 4–47
SET FETCH command, 4–48
SET LOG command, 4–50
  compared with SET OUTPUT command, 4–50
SET NOLOG command, 4–50
SET OUTPUT command, 4–51
  compared with SET LOG command, 4–50
  /HEADER qualifier, 4–51
  /INDEX qualifier, 4–51
  /NOHEADER qualifier, 4–51
  /NOINDEX qualifier, 4–51
SET OUTPUT command/SINGLE_COMMAND qualifier, 4–51
SET PROCESS command, 2–13, 4–53
  /ADDRESS qualifier, 4–53
  /ID=nn qualifier, 4–53
  /NEXT qualifier, 4–53
  /SYSTEM qualifier, 4–53
SET RMS command, 4–56
SET SIGN_EXTEND command, 4–59
SET SYMBOLIZE command, 4–60
SFSBs (shared file synchronization blocks), 4–57
Shadow set displaying SDA information, 4–83
Shareable address data section, 4–29
Shared file synchronization blocks
  See SFSBs
SHOW ADDRESS command, 4–61
SHOW BUGCHECK command, 4–63
SHOW CALL FRAME command, 4–64
SHOW CLUSTER command, 4–66
  /ADDRESS qualifier, 4–66
  /CSID qualifier, 4–66
  /NODE qualifier, 4–66
  /SCS qualifier, 4–66
SHOW CONNECTIONS command, 4–72
  /ADDRESS qualifier, 4–72
  /NODE qualifier, 4–72
  /SYAP qualifier, 4–72
SHOW CPU command, 2–13, 4–45, 4–74
  analyzing a running system, 2–12
SHOW CRASH command, 2–13, 2–22, 2–23, 4–45, 4–77
  analyzing a running system, 2–12
  /CPU qualifier, 4–77
SHOW DEVICE command, 2–22, 4–81
  /ADDRESS qualifier, 4–81
  /CDT qualifier, 4–81
SHOW DUMP command, 4–85
  /ALL qualifier, 4–85
  /BLOCK qualifier, 4–85
  /COMPRESSION_MAP qualifier, 4–85
  /ERROR_LOGS qualifier, 4–85
  /HEADER qualifier, 4–85
  /LMB qualifier, 4–85
  /MEMORY_MAP qualifier, 4–86
  /SUMMARY qualifier, 4–86
SHOW EXECUTIVE command, 4–88
  /SUMMARY qualifier, 4–88
SHOW GALAXY command, 4–92
SHOW GCT command, 4–93
  /ADDRESS qualifier, 4–93
  /ALL qualifier, 4–93
  /CHILDREN qualifier, 4–93
  /HANDLE qualifier, 4–93
  /OWNER qualifier, 4–93
  /SUMMARY qualifier, 4–93
  /type qualifier, 4–93
SHOW GLOBAL SECTION TABLE command, 4–96
  /SECTION_INDEX qualifier, 4–96
SHOW GLOCK command, 4–98
  /ADDRESS qualifier, 4–98
  /ALL qualifier, 4–98
  /BRIEF qualifier, 4–98
  /GMDB_TABLE qualifier, 4–98
  /HANDLE qualifier, 4–98
  /PROCESS_TABLE qualifier, 4–98
  /SYSTEM_TABLE, 4–98
SHOW GMDB command, 4–101
  / ADDRESS qualifier, 4–101
  / ALL qualifier, 4–101
  / NODE qualifier, 4–101
  / SUMMARY qualifier, 4–101
SHOW GSD command, 4–103
  /ADDRESS qualifier, 4–103
  /ALL qualifier, 4–103
  /DELETED qualifier, 4–103
  /GLXGRP qualifier, 4–103
  /GLXSYS qualifier, 4–103
  /GROUP qualifier, 4–103
  /SYSTEM qualifier, 4–103
SHOW GST command, 4–105
SHOW HEADER command, 4–106
SHOW LAN command, 4–106
  /CLIENT qualifier, 4–106
  /CLUEXIT qualifier, 4–106
  /COUNTERS qualifier, 4–106
  /CSMACD qualifier, 4–106
SHOW LAN command (cont’d)
/DEVICE qualifier, 4–106
/ELAN qualifier, 4–107
/ERRORS qualifier, 4–107
/FDDI qualifier, 4–107
/FULL qualifier, 4–107
/COUNTERS qualifier, 4–107
/QUEUE qualifier, 4–107
/UNIT qualifier, 4–107
/VCI qualifier, 4–107

SHOW LOCK command
/ADDRESS qualifier, 4–116
/ALL qualifier, 4–116
/BLOCKING qualifier, 4–116
/BRIEF qualifier, 4–116
/CACHED qualifier, 4–116
/CONVERT qualifier, 4–116
/GRANTED, 4–116
/NAME qualifier, 4–116
/POOL qualifier, 4–116
/SUMMARY qualifier, 4–117
/WAITING qualifier, 4–117

SHOW LOCKS command, 4–116
/STATUS qualifier, 4–117

SHOW MACHINE_CHECK command, 2–13, 4–121
/FULL qualifier, 4–121

SHOW MEMORY command, 2–6, 4–123
/ALL qualifier, 4–123
/BUFFER_OBJECTS qualifier, 4–123
/CACHE qualifier, 4–123
/FILES qualifier, 4–123
/FULL qualifier, 4–123
/GH_REGIONS qualifier, 4–123
/PHYSICAL_PAGES qualifier, 4–124
/POOL qualifier, 4–124
/RESERVED qualifier, 4–124
/SLOTS qualifier, 4–124

SHOW PAGE_TABLE command, 4–125
/FREE qualifier, 4–125
/GLOBAL qualifier, 4–125
/GPT qualifier, 4–125
/INVALID_PRN qualifier, 4–126
/L1 qualifier, 4–126
/L2 qualifier, 4–126
/L qualifier, 4–126
/NONMEMORY qualifier, 4–126
/PTE_ADDRESS qualifier, 4–126
/PT qualifier, 4–126
/SOS1 qualifier, 4–126
/S2 qualifier, 4–126
/SECTION_INDEX qualifier, 4–126
/SPTW qualifier, 4–126

SHOW PAGE_TABLE command/HEADER
/qualifier, 4–125

SHOW PARAMETER command, 4–131
/ACP qualifier, 4–131
/ALL qualifier, 4–131
/CLUSTER qualifier, 4–131
/DYNAMIC qualifier, 4–131
/GALAXY qualifier, 4–131
/GEN qualifier, 4–131
/JOB qualifier, 4–131
/LGI qualifier, 4–131
/MAJOR qualifier, 4–131
/MULTIPROCESSING qualifier, 4–131
/PQL qualifier, 4–131
/RMS qualifier, 4–132
/SCS qualifier, 4–132
/SPECIAL qualifier, 4–132
/STARTUP qualifier, 4–132
/SYS qualifier, 4–132
/TTY qualifier, 4–132

SHOW PFN_DATA command, 4–134
/ADDRESS qualifier, 4–134
/ALL qualifier, 4–134
/BAD qualifier, 4–134
/COLOR qualifier, 4–135
/FREE qualifier, 4–135
/MAP qualifier, 4–135
/MODIFIED qualifier, 4–135
/PRIVATE qualifier, 4–135
/RAD qualifier, 4–135
/UNTESTED qualifier, 4–136
/ZERO qualifier, 4–136

SHOW POOL command, 4–139
/ALL qualifier, 4–139
/BAP qualifier, 4–139
/BRIEF qualifier, 4–139
/CHECK qualifier, 4–139
/FREE qualifier, 4–139
/HEADER qualifier, 4–139
/MAXIMUM_BYTES qualifier, 4–139
/NONPAGED qualifier, 4–139
/PAGED qualifier, 4–140
/RING_BUFFER qualifier, 4–140
/STATISTICS qualifier, 4–140
/SUBTYPE qualifier, 4–140
/SUMMARY qualifier, 4–140
/TYP descriptor qualifier, 4–140

SHOW PORTS command
/ADDRESS qualifier, 4–146
/BUS qualifier, 4–146
/CHANNEL qualifier, 4–146
/DEVICE qualifier, 4–146
/MESSAGE qualifier, 4–146
/NODE qualifier, 4–146
/VC qualifier, 4–146

SHOW PRN_DATA command
/SYSTEM qualifier, 4–136
SHOW PROCESS/ALL command, 4–156
SHOW PROCESS/LOCKS command, 4–118
SHOW PROCESS/RMS command, 4–182
  selecting display options, 4–57
SHOW PROCESS command, 4–54, 4–150
 /ADDRESS qualifier, 4–150
 /ALL qualifier, 4–150
 /AUTHORIZED qualifier, 4–151
 /BRIEF qualifier, 4–151
 /BUFFER_OBJECTS qualifier, 4–151
 /CHANNEL qualifier, 4–151
 /FANDLES qualifier, 4–151
 /FID_ONLY qualifier, 4–151
 /GSTX qualifier, 4–151
 /ID qualifier, 4–151
 /IMAGES qualifier, 4–151
 /INDEX qualifier, 4–151
 /INVALID_PFN qualifier, 4–152
 /LOCKS qualifier, 4–152
 /NEXT qualifier, 4–152
 /NONMEMORY_PFN qualifier, 4–152
 /PAGE_TABLES qualifier, 4–152
 /PCB qualifier, 4–153
 /PERSONA/RIGHTS/AUTHORIZED qualifier, 4–154
 /PERSONA/RIGHTS qualifier, 4–154
 /PERSONA qualifier, 4–153
 /PHD qualifier, 4–154
 /PT qualifier, 4–154
 /PROCESS_SECTION_TABLE qualifier, 4–154
 /PST qualifier, 4–154
 /PTE_ADDRESS qualifier, 4–154
 /PT qualifier, 4–154
 /RDE qualifier, 4–154
 /REGIONS qualifier, 4–154
 /REGISTERS qualifier, 4–154
 /RIGHTS qualifier, 4–154
 /RMS qualifier, 4–155
 /SECTION_INDEX qualifier, 4–155
 /SEMAPHORE qualifier, 4–155
 /SYSTEM qualifier, 4–155
 /THREADS qualifier, 4–155
 /WORKING SET LIST qualifier, 4–156
SHOW RAD command, /ALL qualifier, 4–173
SHOW RESOURCES command, 4–118, 4–175
 /ADDRESS qualifier, 4–175
 /ALL qualifier, 4–175
 /BRIEF qualifier, 4–175
 /CACHED qualifier, 4–175
 /CONTENTION qualifier, 4–175
 /LOCKID qualifier, 4–175
 /NAME qualifier, 4–175
 /OWNED qualifier, 4–175
 /STATUS qualifier, 4–176
SHOW RMD command, 4–180
 /ADDRESS qualifier, 4–180
 /ALL qualifier, 4–180
SHOW RMS command, 4–182
SHOW RSPID command, /CONNECTION qualifier, 4–183
SHOW SHM_CPP command, 4–185
 /ADDRESS qualifier, 4–185
 /ALL qualifier, 4–185
 /IDENT qualifier, 4–185
 /PFN qualifier, 4–185
SHOW SHM_REG command, 4–188
 /ADDRESS qualifier, 4–188
 /ALL qualifier, 4–188
 /IDENT qualifier, 4–188
SHOW SPINLOCKS command, 4–191
 /ADDRESS qualifier, 4–190
 /BRIEF qualifier, 4–190
 /COUNTS qualifier, 4–190
 /DYNAMIC qualifier, 4–190
 /FULL qualifier, 4–190
 /INDEX qualifier, 4–190
 /OWNED qualifier, 4–190
 /STATIC qualifier, 4–191
SHOW STACK command, 4–196
 /ALL qualifier, 4–196
 /EXECUTIVE qualifier, 4–196
 /interrupt qualifier, 4–196
 /KERNEL qualifier, 4–196
 /LONG qualifier, 4–196
 /PHYSICAL qualifier, 4–196
 /QUAD qualifier, 4–196
 /SUPERVISOR qualifier, 4–196
 /SYSTEM qualifier, 4–196
 /USER qualifier, 4–196
SHOW SUMMARY command, 4–150, 4–200
 /IMAGE qualifier, 4–200
 /PROCESS_NAME qualifier, 4–200
 /THREAD qualifier, 4–200
 /USER qualifier, 4–200
 /VALUE qualifier, 4–203
SHOW SYMBOL command, 4–203
 /ALL qualifier, 4–203
 /ALPHA qualifier, 4–203
SHOW TQE command, 4–205
 /ADDRESS qualifier, 4–205
 /ALL qualifier, 4–205
 /BACKLINK qualifier, 4–205
 /PID qualifier, 4–205
 /ROUTINE qualifier, 4–205
SHOW WORKING SET LIST command, 4–207
Signal array, 2–25
SISR register, displaying, 4–75
SISR symbol, 2–19
Site-specific startup command procedure, 2–8, 5–15
  releasing page file blocks, 2–8
Software interrupt status register, 2–19
SPAWN command, 4–208
 /INPUT qualifier, 4–208
 /NOLOGICAL_NAMES qualifier, 4–208
SPAWN command (cont’d)
/NOSYMBOLS qualifier, 4–208
/NOTIFY qualifier, 4–208
/NOWAIT qualifier, 4–208
/OUTPUT qualifier, 4–208
/PROCESS qualifier, 4–209
Spin locks
displaying SDA information, 4–190
owned, 4–75
Spinlock tracing, 6–1
Spinlock Tracing utility, using, 6–2
SPL$DEBUG.EXE file, 4–36
SPL LOAD command, 6–4
SPL SHOW COLLECT command, 6–5
SPL SHOW TRACE
/NOWAIT qualifier, 6–6
/WAIT qualifier, 6–6
SPL SHOW TRACE command, 6–6
/ACQUIRE qualifier, 6–6
/FORKLOCK qualifier, 6–6
/FRKDSPTH qualifier, 6–6
/FRKEND qualifier, 6–6
/NOACQUIRE qualifier, 6–6
/NOFORKLOCK qualifier, 6–6
/NOFRKDSPTH qualifier, 6–6
/NOFRKEND qualifier, 6–6
/NORELEASE qualifier, 6–6
/NOSPINLOCK qualifier, 6–6
/RELEASE qualifier, 6–6
/SPINLOCK qualifier, 6–6
/SUMMARY qualifier, 6–7
/TOP qualifier, 6–7
SPL START COLLECT command
/ADDRESS qualifier, 6–11
/SPINLOCK qualifier, 6–11
SPL START TRACE
/FORKLOCK qualifier, 6–12
/FRKDSPTH qualifier, 6–12
/NOFORKLOCK qualifier, 6–12
/NOFRKDSPTH qualifier, 6–12
SPL START TRACE command
/ACQUIRE qualifier, 6–12
/BUFFER qualifier, 6–12
/CPU qualifier, 6–13
/FRKEND qualifier, 6–13
/NOACQUIRE qualifier, 6–12
/NOFRKEND qualifier, 6–13
/NORELEASE qualifier, 6–12
/NOSPINLOCK qualifier, 6–12
/NOWAIT qualifier, 6–12
/RELEASE qualifier, 6–12
/SPINLOCK qualifier, 6–12
/WAIT qualifier, 6–12
SPL STOP COLLECT command, 6–14
SPL STOP TRACE command, 6–15
SPL UNLOAD command, 6–16
SP symbol, 2–19
SPTs (system page tables), displaying, 4–125
SPTs (system page tables), in system dump file, 2–5
SSPI.EXE file, 4–36
SSP symbol, 2–19
SSRVEXCEPT bugcheck, 2–23
Stack frames
displaying in SDA, 4–64
following a chain, 4–64
Stacks displaying contents, 4–196
Start I/O routine, 4–82
Subprocesses, 4–208
Subtraction operator (−), 2–16
Supervisor stack
displaying contents, 4–196
pointer to, 2–19
Symbols
defining
for SDA, 4–7
evaluating, 4–203
listing, 4–203
loading into the SDA symbol table, 4–34
name, 4–7
representing executive modules, 4–89
user-defined, 4–7
Symbol table files reading into SDA symbol table, 4–35
Symbol tables, specifying an alternate SDA, 3–6
SYS$ATMWORKS351.EXE file, 4–37
SYS$CLUSTER.EXE file, 4–37
SYS$DISK
as SDA output, 4–51
global read, 4–35
SYS$EW1000A.EXE file, 4–37
SYS$GALAXY.STB file, 4–37
SYS$IPC_SERVICES.EXE file, 4–37
SYS$LAN.EXE file, 4–37
SYS$LAN_ATM.EXE file, 4–37
SYS$LAN_ATM4.EXE file, 4–37
SYS$LAN_CSMACD.EXE file, 4–37
SYS$LAN_FDDI.EXE file, 4–37
SYS$LAN_TR.EXE file, 4–37
SYS$LOADABLE IMAGES:SYS.EXE file contents, 4–35
SYS$MME_SERVICES.STB file, 4–37
SYS$NTA.STB file, 4–37
SYS$QIOSERVER_KCLIENT.EXE file, 4–37
SYS$QIOSERVER_KSERVER.EXE file, 4–37
SYS$SCS.EXE file, 4–37
SYS$SYSTEM:PAGEFILE.SYS file, 2–34
SYS$SYSTEM:SYS.EXE file, 4–33
See also System dump files
as dump file, 2–7
releasing blocks containing a crash dump, 3–5
SY$SYSTEM:SYS.EXE file, 4–33
contents, 4–88
SYS$SYSTEM:SYSDEFS.TB file, 2-12
SYS$SYSTEM:SYSDUMP.DMP file, 2-34
  See also System dump files
  protection, 2-8
  size of, 2-6
SYS$TRANSACTION_SERVICES.EXE file, 4-37
SYS$UTC_SERVICES.EXE file, 4-37
SYS$XFACACHE*.STB file, 4-37
SYSAP (system application), 4-183
SYSDEVICE.STB file, global symbols, 4-37
SYSGETSYI.STB file, global symbols, 4-37
SYSLDR_DYN.STB file, global symbols, 4-37
SYSLICENSE.STB file, global symbols, 4-37
System blocks
  See SBs
System Code Debugger, 1-1, 8-1
  interface options, 8-2
  networking, 8-9
  starting, 8-7
System Code Debugger, sample session, 8-14
System Code Debugger commands, 8-8
System Communications Services
  See SCS
System control block base register, 2-19
System Dump Analyzer (SDA) commands, 8-8
System Dump Analyzer utility (SDA), 1-1
  invoked automatically on reboot, 5-1
System Dump Debugger, 1-2, 9-1
  access to symbols in
    OpenVMS executive images, 9-6
    limitations, 9-6
  preparing a System Dump, 9-2
  sample session, 9-6
  setting up test system, 9-3
  setting up the build system, 9-3
  starting, 9-4
  summary of commands, 9-4
  user-interface options, 9-1
  using commands, 9-5
System dump files, 2-3 to 2-7
  mapping physical memory to, 2-10
  requirements for analysis, 2-10
System failures
  analyzing, 2-21
  causing, 2-33 to 2-35
  diagnosing from PC contents, 2-22
  summary, 4-77
System hang, 2-33
System images
  contents, 4-35, 4-88
  global symbols, 4-33
System management creating a crash dump file, 2-3
System message routines global symbols, 4-36
System page file
  as dump file, 2-7
  releasing blocks containing a crash dump, 3-5
System page tables
  See SPTs
System PCBs (process control blocks) displaying, 4-155
System processes, 4-53
System region, examining, 4-19
Systems
  analyzing running, 2-2, 2-12, 3-1
  investigating performance problems, 2-12
System space base address, 2-18
System space operator (G), 2-15
System symbol table, 2-10
System time quadword examining, 4-20
SYSTEM_DEBUG.EXE file, 4-38
SYSTEM_PRIMITIVES.STB file, global symbols, 4-38
SYSTEM_SYNCHRONIZATION_xxx.STB file, global symbols, 4-38
T
  TCP$BGDRIVER.STB file, global symbols, 4-38
  TCP$INETETACP.STB file, global symbols, 4-38
  TCP$INTERNET_SERVICES.STB file, global symbols, 4-38
  TCPIP$NET_GLOBALS.STB file, 2-18
  TCPIP$NFS_GLOBALS.STB file, 2-18
  TCPIP$NFS_SERVICES.STB file, 4-38
  TCPIP$PROXY_GLOBALS.STB file, 2-18
  TCPIP$PROXY_SERVICES.STB file, 4-38
  TCPIP$PWIPACCP.STB file, global symbols, 4-38
  TCPIP$PWIPD lacked.STB file, global symbols, 4-38
  TCPIP$PWIP_GLOBALS.STB file, 2-18
  TCPIP$TNDRIVER.STB file, global symbols, 4-38
  TCPIP$TN_GLOBALS.STB file, 2-18
  Terminal keys defining for SDA, 4-9
  TMSCCP.EXE file, 4-38
  Trace table entries
    See WPPTEs
  Transaction processing, global symbols, 2-18
U
  UCBs (unit control blocks), 4-72
  Unary operators, 2-15 to 2-16
  UNDEFINE command, 4-210
  Unit control blocks
    See UCBs
  UNXSIGNAL bugcheck, 2-23
  Use process-permanent I/O data structures
    See PIO
  User stacks
    displaying contents, 4-196
    pointer, 2-19
  Using the $QIO interface Watchpoint utility, 10-3
Using the Spinlock Tracing utility, 6–2
USP symbol, 2–19

V

VALIDATE PFN_LIST command, 4–211
/ALL qualifier, 4–211
/BAD qualifier, 4–211
/FREE qualifier, 4–211
/MODIFIED qualifier, 4–211
/PRIVATE qualifier, 4–211
/UNTESTED qualifier, 4–211
/ZERO qualifier, 4–211
VALIDATE QUEUE command, 4–213
/BACKLINK qualifier, 4–213
/LIST qualifier, 4–213
/PHYSICAL qualifier, 4–213
/QUADWORD qualifier, 4–213
/SELF_RELATIVE qualifier, 4–213
/SINGLY_LINKED qualifier, 4–213
VALIDATE SHM_CPP command, 4–215
/ADDRESS qualifier, 4–215
/ALL qualifier, 4–215
/IDENT qualifier, 4–215
/PRN qualifier, 4–215
VCBs (volume control blocks), 4–83
Virtual address operator (^V), 2–15
VMS_EXTENSION.EXE file, 4–38
Volume control blocks
See VCBs
Votes, 4–66

VPTB register, displaying, 4–75

W

Watchpoint control blocks
See WPCBs
Watchpoint protection, 10–9
Watchpoint restore entries
See WPREs
Watchpoint restrictions, 10–10
Watchpoint utility, 1–2
Watchpoint utility (WP)
implementation, 10–1
WCBs (window control blocks), 4–57
Window control blocks
See WCBs
WPCBs (watchpoint control blocks), 10–6
WPDRIVER
data structures, 10–6
device driver, 10–1
invoking, 10–5
WPPTEs (trace table entries), 10–7
WPREs (watchpoint restore entries), 10–6

X

XABs (extended attribute blocks), 4–57
XDELTA breakpoint, 10–3
XOR operator (^), 2–16