OpenVMS VAX System Dump Analyzer Utility Manual

Order Number: AA–PV6TD–TE

April 2001

This manual explains how to use the System Dump Analyzer (SDA) to investigate system failures and examine a running system.

Revision/Update Information: This manual supersedes the *VMS System Dump Analyzer Utility Manual*, Version 6.0

Software Version: OpenVMS VAX Version 7.3

Compaq Computer Corporation
Houston, Texas

ZK4556

The Compaq *OpenVMS* documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

Contents

Preface vii

SDA Description SDA–1

1 System Management and SDA SDA–4
1.1 Understanding the System Dump File SDA–4
1.1.1 Choosing Between SYSDUMP.DMP and PAGEFILE.SYS Files . . . SDA–4
1.1.2 Choosing a Dump File Style SDA–6
1.2 Saving System Dumps SDA–6
1.3 Invoking SDA in the Site-Specific Startup Command Procedure SDA–7
2 Analyzing a System Dump SDA–8
2.1 Invoking SDA SDA–9
2.2 Mapping the Contents of the Dump File SDA–9
2.3 Building the SDA Symbol Table SDA–10
2.4 Executing the SDA Initialization File (SDA$INIT) SDA–10
3 Analyzing a Running System SDA–11
4 SDA Context SDA–12
5 CPU Context SDA–12
6 Process Context SDA–13
7 SDA Command Format SDA–15
7.1 General Command Format SDA–15
7.2 Expressions SDA–15
7.2.1 Radix Operators SDA–16
7.2.2 Arithmetic and Logical Operators SDA–16
7.2.3 Precedence Operators SDA–17
7.2.4 Symbols SDA–17
8 Investigating System Failures SDA–19
8.1 General Procedure for Analyzing System Failures SDA–19
8.2 Fatal Bugcheck Conditions SDA–20
8.2.1 Fatal Exceptions SDA–20
8.2.2 Illegal Page Faults SDA–23
9 A Sample System Failure SDA–24
9.1 Identifying the Bugcheck SDA–25
9.2 Identifying the Exception SDA–25
9.3 Locating the Source of the Exception SDA–26
9.3.1 Finding the Driver by Using the Program Counter SDA–26
9.3.2 Calculating the Offset into the Driver's Program Section SDA–27
9.4 Finding the Problem Within the Routine SDA–28
9.4.1 Examining the Routine SDA–28
9.4.2 Checking the Values of Key Variables SDA–29
9.4.3 Identifying and Correcting the Defective Code SDA–30
10 Inducing a System Failure SDA–31
10.1 Meeting Crash Dump Requirements SDA–31
10.2 Examples of How to Cause System Failures SDA–32

SDA Usage Summary SDA–35
SDA Qualifiers SDA–36
/CRASH_DUMP SDA–37
/RELEASE SDA–38
/SYMBOL SDA–39
/SYSTEM SDA–40
SDA Commands SDA–41
@ (Execute Procedure) SDA–44
ATTACH SDA–45
COPY SDA–46
DEFINE SDA–47
EVALUATE SDA–51
EXAMINE SDA–53
EXIT SDA–57
FORMAT SDA–58
HELP SDA–60
READ SDA–62
REPEAT SDA–67
SEARCH SDA–69
SET CPU SDA–71
SET LOG SDA–74
SET OUTPUT SDA–75
SET PROCESS SDA–76
SET RMS SDA–79
SHOW CALL_FRAME SDA–82
SHOW CLUSTER SDA–85
SHOW CONNECTIONS SDA–90
SHOW CPU SDA–94
SHOW CRASH SDA–98
SHOW DEVICE SDA–103
SHOW EXECUTIVE SDA–110
SHOW HEADER SDA–112
SHOW LAN SDA–113
SHOW LOCK SDA–121
SHOW LOGS SDA–125
SHOW PAGE_TABLE SDA–126
SHOW PFN_DATA SDA–131
SHOW POOL SDA–135
SHOW PORTS SDA–142
SHOW PROCESS SDA–149
SHOW RESOURCE SDA–161
SHOW RMS SDA–166
SHOW RSPID SDA–167
SHOW SPINLOCKS SDA–169
SHOW STACK SDA–176
SHOW SUMMARY SDA–178

SHOW SYMBOL SDA–181
SHOW TRANSACTIONS SDA–182
SPAWN SDA–183
VALIDATE QUEUE SDA–185

Index

Figures

SDA–1 Pointer Argument List on the Stack SDA–21
SDA–2 Mechanism Array SDA–22
SDA–3 Signal Array SDA–22
SDA–4 Stack Following an Illegal Page-Fault Error SDA–24
SDA–5 Call Frame SDA–83

Tables

SDA–1 Selecting and Displaying Information About Processes SDA–1
SDA–2 Displaying Information about Data Structures SDA–2
SDA–3 Examining, Evaluating, and Validating Information SDA–2
SDA–4 Searching for, Formatting, and Copying Information SDA–3
SDA–5 Managing the SDA Utility and the SDA Symbol Table SDA–3
SDA–6 Displaying Information Produced by DECdtm SDA–3
SDA–7 Comparison of Full and Subset Dump Files SDA–6
SDA–8 SDA Operators SDA–16
SDA–9 SDA Symbols SDA–17
SDA–10 Descriptions of SDA Qualifiers SDA–36
SDA–11 Descriptions of SDA Commands SDA–41
SDA–12 Modules Containing Global Symbols and Data Structures Used by SDA SDA–63
SDA–13 Modules Defining Global Locations Within the Executive Image SDA–63
SDA–14 SET RMS Command Keywords for Displaying Process RMS Information SDA–79
SDA–15 Contents of the SHOW LOCK and SHOW PROCESS/LOCKS Displays SDA–122
SDA–16 Virtual Page Information in the SHOW PAGE_TABLE Display SDA–126
SDA–17 Physical Page Information in the SHOW PAGE_TABLE Display SDA–128
SDA–18 Page Frame Number Information in the SHOW PFN_DATA Display SDA–131
SDA–19 Process Section Table Entry Information in the SHOW PROCESS Display SDA–154
SDA–20 Process I/O Channel Information in the SHOW PROCESS Display . . . SDA–155
SDA–21 Resource Information in the SHOW RESOURCE Display SDA–161
SDA–22 Static Spin Locks SDA–170
SDA–23 Process Information in the SHOW SUMMARY Display SDA–178

Preface

Intended Audience

The *OpenVMS VAX System Dump Analyzer Utility Manual* is primarily intended for the system programmer who must investigate the causes of system failures and debug kernel-mode code, such as a device driver. This programmer should have some knowledge of OpenVMS data structures to properly interpret the results of System Dump Analyzer (SDA) commands.

This manual also includes information required by the system manager in order to maintain the system resources necessary to capture and store system crash dumps. Those who need to determine the cause of a hung process or improve system performance can refer to this manual for instructions for using SDA to analyze a running system.

Document Structure

The *OpenVMS VAX System Dump Analyzer Utility Manual* contains the following sections:

Section	Description of Contents
SDA Description	Includes the following information: • An introduction to the functions of the System Dump Analyzer (SDA) • A description of SDA features • A discussion of key concepts of SDA • An illustration of the use of SDA This section also includes instructions for maintaining the optimal environment for the analysis of system failures and notes the requirements for processes invoking SDA.
SDA Usage Summary	Summarizes how to use SDA, including invoking SDA, exiting from SDA, and recording the output of an SDA session. It also describes required privileges.
SDA Qualifiers	Describes ANALYZE command qualifiers that govern the behavior of SDA: /CRASH_DUMP, /RELEASE, /SYMBOL, and /SYSTEM.

Section	Description of Contents
SDA Commands	Describes each SDA command; descriptions include the following information about each command: • Function • Format • Parameters This section also provides examples of situations in which specific commands are useful.

Related Documents

Additional information is available in the following documents:

- *OpenVMS System Manager's Manual, Volume 1: Essentials*
- *OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems*
- *OpenVMS System Management Utilities Reference Manual*
- *Guide to Creating OpenVMS Modular Procedures*
- *OpenVMS Performance Management*
- *OpenVMS VAX Device Support Manual*[1]
- *OpenVMS DCL Dictionary*
- *OpenVMS System Services Reference Manual*

Investigators of VMScluster failures will find the discussion in *OpenVMS Cluster Systems* and the discussion of the Show Cluster utility in the *OpenVMS System Management Utilities Reference Manual* helpful in understanding the output of several SDA commands.

For additional information about Compaq *OpenVMS* products and services, access the Compaq website at the following location:

`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

[1] This manual has been archived but is available on the OpenVMS Documentation CD-ROM.

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

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.
Return	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 choose 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 required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold text	This typeface 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 type 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 decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

SDA Description

When a fatal error causes the system to fail, the operating system copies the contents of memory to a system dump file; the system also records the hardware context of each processor in the system.

The System Dump Analyzer (SDA) provides a means of interpreting the contents of the system dump file, thus enabling you to examine the status of each processor at the time of the failure and to investigate the probable causes of the crash.

To examine the system dump file, you invoke SDA by using the DCL command ANALYZE/CRASH_DUMP. You can also invoke SDA to analyze a running system, using the DCL command ANALYZE/SYSTEM. Most SDA commands generate useful output in this mode of operation.

Caution

Although the analysis of a running system might be instructive, be aware that system context, process context, and a processor's hardware context remain fluid during any given display. In a multiprocessing environment, a process running SDA might be rescheduled to a different processor frequently during analysis. Therefore, Compaq recommends that you not examine the hardware context of processors in a running system.

Following are brief explanations of SDA qualifiers. Details about these qualifiers are in the SDA Qualifiers section.

Qualifier	Description
/CRASH_DUMP	Invokes SDA to analyze a specified dump file
/RELEASE	Invokes SDA to release those blocks that are occupied by a crash dump in a specified system paging file
/SYMBOL	Specifies a system symbol table for SDA to use in place of the system symbol table it uses by default (SYS$SYSTEM:SYS.STB)
/SYSTEM	Invokes SDA to analyze a running system

The following tables show the SDA commands that you can use to perform operations within the SDA utility. These commands are in groups of related information. Details about SDA commands are in the SDA Commands section.

Table SDA–1 describes information that you can select and display about processes.

Table SDA–1 Selecting and Displaying Information About Processes

Operation	SDA Command
Display the condition of the operating system and the hardware context of each processor in the system at the time of a crash	SHOW CRASH
Display a summary of all processes on the system	SHOW SUMMARY

(continued on next page)

Table SDA–1 (Cont.) Selecting and Displaying Information About Processes

Operation	SDA Command
Select a process to become the SDA current process	SET PROCESS
Examine the memory of any process	SHOW PROCESS
Select a specific processor in a multiprocessing system as the subject of analysis	SET CPU
Display information about the state of a processor at the time of the system failure	SHOW CPU
Display multiprocessor synchronization information	SHOW SPINLOCKS
Display the contents of a specific process stack or the interrupt stack of a specific processor	SHOW STACK
Display the layout of the loadable executive images	SHOW EXECUTIVE

Table SDA–2 describes information that you can display about data structures.

Table SDA–2 Displaying Information about Data Structures

Operation	SDA Command
Display memory management data structures	SHOW POOL, SHOW PFN_DATA, SHOW PAGE_TABLE
Display device status, as reflected in system data structures	SHOW DEVICE
Display OpenVMS RMS data structures of a process	SHOW PROCESS/RMS
Display lock management data structures	SHOW RESOURCE, SHOW LOCK
Display information contained in various local area network (LAN) data structures	SHOW LAN
Display VAXcluster management data structures	SHOW CLUSTER, SHOW CONNECTIONS, SHOW RSPID, SHOW PORTS

Table SDA–3 describes SDA commands that you can use to examine, evaluate, and validate information.

Table SDA–3 Examining, Evaluating, and Validating Information

Operation	SDA Command
Evaluate an expression in hexadecimal and decimal, interpreting its value as a symbol, a condition value, a page table entry (PTE), or a processor status longword (PSL)	EVALUATE
Examine the contents of memory locations, optionally interpreting them as MACRO instructions, a PTE, or a PSL	EXAMINE
Validate the integrity of the links in a queue	VALIDATE QUEUE

Table SDA–4 describes the SDA commands that you can use to search for, format, and copy information.

Table SDA–4 Searching for, Formatting, and Copying Information

Operation	SDA Command
Search memory for a given value	SEARCH
Format system data structures	FORMAT
Format a call frame from a stack location	SHOW CALL_FRAME
Copy the system dump file	COPY

Table SDA–5 describes the operations you can perform to manage the SDA utility and the SDA symbol table.

Table SDA–5 Managing the SDA Utility and the SDA Symbol Table

Operation	SDA Command
Define keys to invoke SDA commands	DEFINE/KEY
Switch control of your terminal from your current process to another process in your job	ATTACH
Direct (or echo) the output of an SDA session to a file or device	SET OUTPUT or SET LOG
Repeat execution of the last command issued	REPEAT
Create a subprocess of the process currently running SDA	SPAWN
Change the options shown by the SHOW PROCESS/RMS command	SET RMS
Define symbols to represent values or locations in memory and add them to the SDA symbol table	DEFINE
Read a set of global symbols into the SDA symbol table	READ
Display the hexadecimal value of a symbol and, if the value is equal to an address location, the contents of that location	SHOW SYMBOL
Exit from the SDA display or from the SDA utility	EXIT

Table SDA–6 describes the commands that you can use to display information produced by DECdtm.

Table SDA–6 Displaying Information Produced by DECdtm

Operation	SDA Command
Display information about all transactions on the node or about a specified transaction	SHOW TRANSACTIONS
Display information about transaction logs currently open for the node	SHOW LOGS

Although SDA provides a great deal of information, it does not 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 send Compaq a Software Performance Report (SPR) and a copy of the system dump file written at the time of the failure.

1 System Management and SDA

The system manager must perform the following operations in regard to the system dump file:

- Ensure that the system writes a dump file whenever the system fails.
- Ensure that the dump file is large enough to contain all the information to be saved.
- Ensure that the dump file is saved for analysis.

The following sections describe these tasks.

1.1 Understanding the System Dump File

The operating system attempts to write information into the system dump file only if the system parameter DUMPBUG is set. [1] If DUMPBUG is set and the operating system fails, the system writes the contents of the error log buffers, processor registers, and physical memory into the system dump file, overwriting its previous contents.

If the system dump file is too small, it cannot contain all of memory when a system failure occurs. For most systems, this means that the system's page table (SPT) is not included in the dump. SDA cannot analyze a dump unless the entire SPT is included in the dump.

1.1.1 Choosing Between SYSDUMP.DMP and PAGEFILE.SYS Files

SYS$SYSTEM:SYSDUMP.DMP (SYS$SPECIFIC:[SYSEXE]SYSDUMP.DMP) is furnished as an empty file in the software distribution kit. To successfully store a crash dump, you must make SYS$SYSTEM:SYSDUMP.DMP large enough to hold all the information to be written when the system fails. If this is not possible, you can have dumps written into the system paging file, SYS$SYSTEM:PAGEFILE.SYS. You can enlarge or adjust the size of either of these files by using the CREATE command of the System Generation utility (SYSGEN), as described in the *OpenVMS System Management Utilities Reference Manual*.

Using SYSDUMP.DMP

To calculate the correct size for SYS$SYSTEM:SYSDUMP.DMP, use the following formula:

```
size-in-blocks(SYS$SYSTEM:SYSDUMP.DMP)
        = size-in-pages(physical-memory)
        + (number-of-error-log-buffers * blocks-per-buffer)
        + 1
```

You can use the DCL command SHOW MEMORY to determine the total size of physical memory on your system. In addition, you must account for any MA780 multiport memory installed on your system. The number of error log buffers in any given system varies, depending on the setting of the ERRORLOGBUFFERS system parameter. (See the *OpenVMS System Management Utilities Reference Manual* for additional information about this parameter.)

[1] The DUMPBUG parameter is set by default. To examine or change its value, consult the *OpenVMS System Management Utilities Reference Manual*.

Using PAGEFILE.SYS

If SYS$SYSTEM:SYSDUMP.DMP does not exist, the operating system writes the dump of physical memory into SYS$SYSTEM:PAGEFILE.SYS, the system's paging 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. If it is clear, the entire paging file is used for paging, and any dump written to the paging file is lost.[2]

Do not use a selective dump (DUMPSTYLE=1) style with PAGEFILE.SYS. If the PAGEFILE is used for a selective dump, and if the PAGEFILE is not large enough to contain all the logical memory blocks, the dump fills the entire pagefile and the system may hang on reboot. When selective dumping is setup, all available space will be used to write out the logical memory blocks. If the pagefile is large enough to contain all of physical memory, there is no reason to use selective dumping and a full memory dump (DUMPSTYLE=0) should be used.

To calculate the minimum size for SYS$SYSTEM:PAGEFILE.SYS, use the following formula:

```
size-in-blocks(SYS$SYSTEM:PAGEFILE.SYS)
        = size-in-pages(physical-memory)
        + (number-of-error-log-buffers * blocks-per-buffer)
        + 1
        + 1000
```

Caution

This formula calculates only the minimum size requirement for saving a dump in the system's primary page file. For most systems, the page file must be larger than this to avoid hanging the system. (See the *OpenVMS System Manager's Manual, Volume 1: Essentials* and *OpenVMS System Manager's Manual, Volume 2: Tuning, Monitoring, and Complex Systems* for more information.)

Freeing Space in PAGEFILE.SYS

If you use SYS$SYSTEM:PAGEFILE.SYS to hold system crash dumps, you must later free the space occupied by the dump so that the pager can use it. Usually, you include SDA commands in the site-specific startup command procedure (SYS$MANAGER:SYSTARTUP_VMS.COM) to free this space; if you do not, your system might hang during the startup procedure.

A common method of freeing space is to copy the dump from SYS$SYSTEM:PAGEFILE.SYS to another file, using the SDA COPY command. (Although you can also use the DCL COPY command to copy a dump file, only the SDA COPY command frees the pages occupied by the dump from the system's paging file.)

Occasionally, you might want to free the pages in the paging file that are taken up by the dump without having to copy the dump elsewhere. When you issue the ANALYZE/CRASH_DUMP/RELEASE command, SDA immediately releases the pages to be used for system paging, effectively deleting the dump.

[2] The SAVEDUMP parameter is clear by default. To examine or change its value, consult the *OpenVMS System Management Utilities Reference Manual*.

Note

The ANALYZE/CRASH_DUMP/RELEASE command does not allow you to analyze the dump before deleting it.

1.1.2 Choosing a Dump File Style

In certain system configurations, it might 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 might not be able to supply enough disk space for a full memory dump. In normal circumstances, if the system dump file cannot accommodate all of memory, SDA cannot analyze the dump.

To preserve those portions of memory that contain information most useful in determining the causes of system failures, a system manager sets the static system parameter DUMPSTYLE to 1. When the DUMPSTYLE parameter is set, AUTOGEN attempts to create a dump file large enough to contain ample information for SDA to analyze a failure. When the DUMPSTYLE parameter is clear (the default), AUTOGEN attempts to create a dump file large enough to contain all of physical memory.

A comparison of full and subset style dump files appears in Table SDA–7.

Table SDA–7 Comparison of Full and Subset Dump Files

	Full	Subset
Available Information	Complete contents of physical memory in use, stored in order of increasing physical address (for instance, system and global page tables are stored last).	System page table, global page table, system space memory, and process and control regions (plus global pages) for all saved processes.
Unavailable Information	Contents of paged-out memory at the time of the crash.	Contents of paged-out memory at the time of the crash, process and control regions of unsaved processes, and memory not mapped by a page table (such as the free and modified lists).
SDA Command Limitations	None.	The following commands are not useful for unsaved processes: SHOW PROCESS/CHANNELS, SHOW PROCESS/RMS, SHOW STACK, and SHOW SUMMARY/IMAGE.

1.2 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. For this reason, as system manager, you need to save the contents of the file after a system failure has occurred.

Using the SDA COPY Command

You can use the SDA COPY command or the DCL COPY command in your site-specific startup procedure. Compaq recommends using the SDA COPY command because it marks the dump file as copied. This is particularly important if the dump was written into the paging file, SYS$SYSTEM:PAGEFILE.SYS, because the SDA COPY command releases to the pager the pages that were occupied by the dump.

Using /IGNORE=NOBACKUP

Because system dump files are set to NOBACKUP, the Backup utility (BACKUP) does not copy dump files 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, the new file is not set to NOBACKUP.

As included in the distribution kit, SYS$SYSTEM:SYSDUMP.DMP is protected against world access. Because a dump file can contain privileged information, Compaq recommends that you continue to protect dump files from universal read access.

1.3 Invoking SDA in the Site-Specific Startup Command Procedure

Because a listing of the SDA output is an important source of information in determining the cause of a system failure, it is a good idea to have SDA produce such a listing after every failure. The system manager can ensure the creation of a listing by modifying the site-specific startup command procedure SYS$MANAGER:SYSTARTUP_VMS.COM so that it invokes SDA when the system is booted.

When invoked in the site-specific startup procedure, SDA executes the specified commands only if the system is booting immediately after a system failure. SDA examines a flag in the dump file's header that indicates whether it has already processed the file. If the flag is set, SDA merely exits. If the flag is clear, SDA executes the specified commands and sets the flag. This flag is clear when the operating system initially writes a crash dump, except for those resulting from an operator-requested shutdown (for instance, SYS$SYSTEM:SHUTDOWN.COM).

Using SYSDUMP.DMP

The following example shows typical commands that you might add to your site-specific startup command procedure to produce an SDA listing after each failure.

```
$ !
$ !      Print dump listing if system just failed
$ !
$ ANALYZE/CRASH_DUMP SYS$SYSTEM:SYSDUMP.DMP
   COPY SYS$SYSTEM:SAVEDUMP.DMP       ! Save dump file
   SET OUTPUT DISK1:SYSDUMP.LIS       ! Create listing file
   READ/EXEC         ! Read symbols into the SDA symbol table
   SHOW CRASH        ! Display crash information
   SHOW STACK        ! Show current stack
   SHOW SUMMARY      ! List all active processes
   SHOW PROCESS/PCB/PHD/REG           ! Display current process
   SHOW SYMBOL/ALL   ! Print system symbol table
   EXIT
$ PRINT DISK1:SYSDUMP.LIS
```

The COPY command in the preceding example saves the contents of the file SYS$SYSTEM:SYSDUMP.DMP. If your system's startup command file does not save a copy of the contents of this file, this crash dump information is lost in the next system failure, when the system saves the information about the new failure, overwriting the contents of SYS$SYSTEM:SYSDUMP.DMP.

Using PAGEFILE.SYS

If you are using the SYS$SYSTEM:PAGEFILE.SYS as the crash dump file, you must include SDA commands in SYS$MANAGER:SYSTARTUP_VMS.COM that free the space occupied by the dump so that the pager can use it. For instance:

```
$ ANALYZE/CRASH_DUMP SYS$SYSTEM:PAGEFILE.SYS
   .
   .
   .
   COPY dump_filespec
   EXIT
```

2 Analyzing a System Dump

SDA performs certain tasks prior to bringing a dump into memory, presenting its initial displays, and accepting command input. This section describes those tasks, which include the following:

- Verifying that the process invoking it has privileges to read the dump file
- Using RMS to read in pages upon request
- Reading the system symbol tables (SYS$SYSTEM:SYS.STB and SYS$SYSTEM:REQSYSDEF.STB)
- Executing the commands in the SDA initialization file

For detailed information about the investigation of a system failure, see Section 8.

Requirements

To be able to analyze a dump file, your process must have the following:

- *Read access* to the file that contains the dump and to copies of the following symbol tables, which SDA reads by default:
  - SYS$SYSTEM:SYS.STB (the system symbol table)
  - SYS$SYSTEM:REQSYSDEF.STB (the required subset of the symbols in the file SYSDEF.STB)
- *A system UIC* or *SYSPRV privilege* for a process to read the dump file.

  As included in the distribution kit, SYS$SYSTEM:SYSDUMP.DMP, SYS$SYSTEM:SYS.STB, and SYS$SYSTEM:REQSYSDEF.STB are protected against world access.
- *Sufficient virtual address space* for SDA to access the entire dump and any required symbol tables.

  To ensure that SDA has the correct amount of virtual address space, a value of 16,000 of the system parameter VIRTUALPAGECNT should be sufficient to analyze any dump, unless there is an exceptionally large number of symbols. You might need to increase the size if your particular installation places heavy demands on the virtual address space of the process.

2.1 Invoking SDA

If your process satisfies these conditions, 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 for the name of the file, as follows:

```
$ ANALYZE/CRASH_DUMP
_Dump File:
```

The default file specification is as follows:

disk:*[default-dir]*SYSDUMP.DMP

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

2.2 Mapping the Contents of the Dump File

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

The SPT appears at the largest physical addresses in a typical configuration. As a result, if a dump file is too small, the SPT cannot be written to it in the event of system failure.

If SDA cannot find the SPT in the dump file, it displays either of the following messages:

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

```
%SDA-E-SHORTDUMP, the dump only contains m out of n pages of physical memory
```

If SDA displays either of these error messages, you cannot analyze the crash dump, but must take steps to ensure that any subsequent dump can be preserved. To do this, you must increase the size of the dump file, as indicated in Section 1.1, or adjust the system DUMPSTYLE parameter, as discussed in Section 1.1.2.

Under certain conditions, the system might not save some memory locations in the system dump file. For instance, during halt/restart bugchecks, the system does not preserve the contents of general registers. If such a bugcheck occurs, SDA indicates in the SHOW CRASH display that the contents of the registers were destroyed. Additionally, if a bugcheck occurs during system initialization, the contents of the register display might be unreliable. The symptom of such a bugcheck is a SHOW SUMMARY display that shows no processes or only the swapper process.

Also, if you use an SDA command to access a virtual address that has no corresponding physical address, SDA displays the following error message:

```
%SDA-E-NOTINPHYS, 'location' not in physical memory
```

When you analyze a subset dump file, if you use an SDA command to access a virtual address that has a corresponding physical address but was not saved in the dump file, SDA displays the following error message:

```
%SDA-E-MEMNOTSVD, memory not saved in the dump file
```

2.3 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. This file, named SYS$SYSTEM:SYS.STB by default, contains most of the global symbols used by the operating system. SDA also reads into its symbol table a subset of SYS$SYSTEM:SYSDEF.STB, called SYS$SYSTEM:REQSYSDEF.STB, that it requires to identify locations in memory.

If SDA cannot find the system symbol table file, or if it is given a file that is not a system symbol table in the /SYMBOL qualifier to the ANALYZE command, it halts with a fatal error.

When SDA finishes building its symbol table, it displays a message identifying itself and the immediate cause of the crash. In the following example, the cause of the crash was an illegal exception occurring at an IPL above IPL$_ASTDEL or while using the interrupt stack.

```
Dump taken on 28-Jan-1993  18:10:09.79
INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
```

2.4 Executing the SDA Initialization File (SDA$INIT)

After displaying the crash 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.

The 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 might 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 SYS$SYSTEM:SYSDEF.STB
```

You might also find it very helpful to define those symbols that identify the modules in the images that make up the executive. You can do this by issuing the following command:

```
READ/EXECUTIVE SYS$LOADABLE_IMAGES
```

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

```
SDA>
```

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

An SDA initialization file can invoke a command procedure with the @ command. However, such command procedures cannot themselves invoke a command procedure (that is, you cannot have nested command procedures).

3 Analyzing a Running System

Occasionally, an internal problem hinders system performance but does not cause a system failure. By allowing you to examine the running system, SDA provides the means to search for the solution to the problem without disturbing the operating system. For example, you can 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 (see *OpenVMS Performance Management*).

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 then does the following:

1. Attempts to load the system symbol table (SYS$SYSTEM:SYS.STB) and symbol table SYS$SYSTEM:REQSYSDEF.STB.
2. Executes the contents of any existing SDA initialization file, as it does when invoked to analyze a crash dump (see Sections 2.3 and 2.4, respectively).
3. Displays its identification message and prompt, as follows:

   ```
   OpenVMS System analyzer

   SDA>
   ```

The SDA> 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 undertaking an analysis of a running system, take the following considerations into account:

- 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, use caution in interpreting its displays. Because system states change frequently, it is possible that the information SDA displays might be inconsistent with the actual, volatile state of the system at any given moment.

- Certain SDA commands are illegal in this mode, such as SHOW CPU and SET CPU. If you use these commands, SDA displays 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. Also, the "Time of system crash" information refers to the time you entered the ANALYZE/SYSTEM command.

4 SDA Context

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

When the subject of analysis is a uniprocessor system, SDA's context is solely **process context**. That is, SDA can interpret its process-specific commands in the context of either the process current on the uniprocessor or some other process in some other scheduling state.

When you initially invoke SDA to analyze a crash dump, its process context defaults to that of the process that was current at the time of the crash. 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.

You can change SDA's process context by issuing any of the following commands:

SET PROCESS/INDEX=**nn**
SET PROCESS **name**
SHOW PROCESS/INDEX=**nn**

5 CPU Context

In a uniprocessor system only one CPU exists, and the concept of SDA CPU context is not an issue. However, for a multiprocessor system with more than one active CPU, SDA must maintain an idea of CPU context to provide a way of displaying information bound to a specific CPU, such as the reason for the bugcheck exception, the currently executing process, the current IPL, the contents of CPU registers, and any owned spin locks. When you first invoke SDA to analyze a crash dump, the "SDA current CPU" is the CPU that induced the system failure.

Changing the CPU Context

You can use several SDA commands to change the CPU context. When you change the CPU context, the "SDA current process" is changed to the current process on the "SDA current CPU" to synchronize CPU context and process context. If no current process is on the "SDA current CPU," the "SDA current process" is undefined; no process context information will be available until you set SDA process context to a specific process.

Type HELP PROCESS_CONTEXT for specific information about the "SDA current process."

The following SDA commands change the "SDA current CPU":

Command	Description
SET CPU cpu_id	Changes the "SDA current CPU" to CPU cpu_id
SHOW CPU cpu_id	Changes the "SDA current CPU" to CPU cpu_id
SHOW CRASH	Changes the "SDA current CPU" to the CPU that induced the system failure

If you select a process that is the current process on a CPU, the following commands change the "SDA current CPU" to that CPU:

SET PROCESS **name**
SET PROCESS/INDEX=**nn**
SHOW PROCESS **name**

SHOW PROCESS/INDEX=**nn**

No other SDA commands affect the "SDA current CPU."

Note

When you analyze the running system, you cannot use the SET CPU and SHOW CPU commands because SDA does not have access to all the CPU-specific information about the running system.

6 Process Context

In a uniprocessor system, process context might be the process that is current on the CPU or the process in whose context process-specific SDA commands are interpreted. For a multiprocessor system with more than one active CPU, however, the meaning of "SDA process context" changes so that it includes a way to display information relevant to a specific process both when the process is current on a processor and when the process is not.

You can use several SDA commands to change SDA process context. Following is a list of the results of some of these changes:

- When you change the "SDA current process" to the current process on a CPU, the "SDA current CPU" is changed to the new CPU to synchronize CPU context and process context.
- When you change the "SDA current process" to a process that is not current on any processor, the "SDA current CPU" is not changed.
- When you change the SDA CPU context to a CPU that has no current process, the "SDA current process" is undefined; no process context information is available until you set SDA process context to a specific process.

Type HELP CPU_CONTEXT for specific information about the "SDA current CPU."

The following SDA commands change the "SDA current process":

Command	Description
SET PROCESS name	Changes the "SDA current process" to the named process
SET PROCESS /INDEX=n	Changes the "SDA current process" to the process with index n
SHOW PROCESS name	Changes the "SDA current process" to the named process
SHOW PROCESS /INDEX=n	Changes the "SDA current process" to the process with index n

The following commands change the SDA process context if the "SDA current process" is not the current process on the selected CPU:

Command	Description
SET CPU cpu_id	Changes the "SDA current process" to the current process on CPU cpu_id
SHOW CPU cpu_id	Changes the "SDA current process" to the current process on CPU cpu_id
SHOW CRASH	Changes the "SDA current process" to the current process on the CPU that induced the system failure

No other SDA commands affect the "SDA current process."

Note

When you analyze the running system, CPU context is not used because all the CPU-specific information might not be available.

Changing the SDA CPU Context

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, such as the reason for the bugcheck exception, the currently executing process, the current IPL, the contents of processor-specific registers, the interrupt stack pointer (ISP), 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.[3]

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

SET CPU **cpu-id**
SHOW CPU **cpu-id**
SHOW CRASH

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 you set SDA process context to that of a specific process.

Likewise, changing process context can involve 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 the **name** or index number (**nn**) refers to a current process:

SET PROCESS **name**
SET PROCESS/INDEX=**nn**
SHOW PROCESS **name**
SHOW PROCESS/INDEX=**nn**

[3] 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.

7 SDA Command Format

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

7.1 General Command Format

SDA uses a command format similar to that used by the DCL interpreter. You issue commands in this general format:

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

where:

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 and SE stands for SET.

/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 a slash must precede each. You can abbreviate qualifiers to the shortest string of characters that uniquely identifies the qualifier.

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 SYS$DISK and default directory are those specified in your most recent SET DEFAULT command. See the *OpenVMS DCL Dictionary* for a description of the DCL command SET DEFAULT.

!comment Consists of text that describes the command, but this text 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.

7.2 Expressions

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

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

The following sections describe elements other than numerals.

7.2.1 Radix Operators

Radix operators determine which numeric base SDA uses to evaluate expressions. You can use one of 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.

7.2.2 Arithmetic and Logical Operators

There are two types of arithmetic and logical operators, both of which are listed in Table SDA–8.

- **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 32-bit operands.

Table SDA–8 SDA Operators

Operator	Action
Unary Operators	
#	Performs a logical NOT of the expression
+	Makes the value of the expression positive
–	Makes the value of the expression negative
@	Evaluates the following expression as a virtual address, then uses the contents of that address as value
G	Adds 80000000_{16} to the value of the expression[1]
H	Adds $7FFE0000_{16}$ to the value of the expression[2]
Binary Operators	
+	Addition
–	Subtraction
*	Multiplication
&	Logical AND
|	Logical OR
\	Logical XOR

[1]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 $80000D40_{16}$.

[2]The unary operator H corresponds to a convenient base address in the control region of a process ($7FFE0000_{16}$). You can therefore refer to an address such as $7FFE2A64_{16}$ as H2A64.

(continued on next page)

Table SDA–8 (Cont.) SDA Operators

Operator	Action
Binary Operators	
/	Division[3]
@	Arithmetic shifting

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

7.2.3 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.

7.2.4 Symbols

Names of symbols can contain from 1 to 31 alphanumeric characters and can include the dollar sign ($) and underscore (_) characters. Symbols can take values from $-7FFFFFFF_{16}$ to $7FFFFFFF_{16}$.

By default, SDA copies symbols into its symbol table from the files SYS$SYSTEM:SYS.STB and SYS$SYSTEM:REQSYSDEF.STB. To add more symbols to the symbol table, you can use the following SDA commands:

- READ—to add symbols from other symbol tables or object modules
- DEFINE—to create symbols and add them to the symbol table

In addition, SDA provides the symbols described in Table SDA–9.

Table SDA–9 SDA Symbols

Symbol	Meaning
. (period)	Current location
2P_CDDB	Address of alternate CDDB for MSCP-served device[1]
2P_UCB	Address of alternate UCB for dual-pathed device[1]
AMB	Associated mailbox UCB pointer[1]
AP	Argument pointer[2]
CDDB	Address of class driver descriptor block for MSCP-served device[1]
CLUSTRLOA	Base address of loadable VAXcluster code
CRB	Address of channel request block[1]
DDB	Address of device data block[1]
DDT	Address of driver dispatch table[1]

[1]The SHOW DEVICE command defines this symbol, if appropriate, to represent information pertinent to the last displayed device unit. See the description of the SHOW DEVICE command for additional information.

[2]The value of those symbols representing the current SDA process context changes whenever you issue a command that changes the context (see Section 4). These symbols include the general-purpose registers (R0 through R11, AP, FP, PC, and SP); the per-process stack pointers (USP, SSP, KSP); the page table base and length registers (P0BR, P0LR, P1BR, and P1LR); and the processor status longword (PSL).

(continued on next page)

Table SDA–9 (Cont.) SDA Symbols

Symbol	Meaning
*nn*DRIVER	Base address of a driver prologue table (DPT); such a symbol exists for each loaded device driver in the system[3]
ESP	Executive stack pointer[2]
FP	Frame pointer[2]
FPEMUL	Base address of the code that emulates floating-point instructions
G	80000000_{16}, the base address of system space
H	$7FFE0000_{16}$
IRP	Address of I/O request packet[1]
JIB	Job information block
KSP	Kernel stack pointer[2]
LNM	Address of logical name block for mailbox[1]
MCHK	Address within loadable CPU-specific routines
MSCP	Address of loadable MSCP server code
ORB	Address of object rights block[1]
P0BR	Base register for the program region (P0)[2]
P0LR	Length register for the program region (P0)[2]
P1BR	Base register for the control region (P1)[2]
P1LR	Length register for the control region (P1)[2]
PC	Program counter[2]
PCB	Process control block
PDT	Address of port descriptor table[1]
PHD	Process header
PSL	Processor status longword[2]
R0 through R11	General registers[2]
RMS	Base address of the RMS image
RWAITCNT	Resource wait count for MSCP-served device[1]
SB	Address of system block[1]
SCSLOA	Base address of loadable common SCS services
SP	Current stack pointer of a process[2]
SSP	Supervisor stack pointer[2]
SYSLOA	Base address of loadable processor-specific system code
TMSCP	Address of loadable TMSCP server code
UCB	Address of unit control block[1]

[1]The SHOW DEVICE command defines this symbol, if appropriate, to represent information pertinent to the last displayed device unit. See the description of the SHOW DEVICE command for additional information.

[2]The value of those symbols representing the current SDA process context changes whenever you issue a command that changes the context (see Section 4). These symbols include the general-purpose registers (R0 through R11, AP, FP, PC, and SP); the per-process stack pointers (USP, SSP, KSP); the page table base and length registers (P0BR, P0LR, P1BR, and P1LR); and the processor status longword (PSL).

[3]The notation *nn* within the symbol *nn*DRIVER represents a 2-letter, generic device/controller name (for example, *LP*DRIVER).

(continued on next page)

Table SDA–9 (Cont.) SDA Symbols

Symbol	Meaning
USP	User stack pointer[2]
VCB	Address of volume control block for mounted device[1]

[1]The SHOW DEVICE command defines this symbol, if appropriate, to represent information pertinent to the last displayed device unit. See the description of the SHOW DEVICE command for additional information.

[2]The value of those symbols representing the current SDA process context changes whenever you issue a command that changes the context (see Section 4). These symbols include the general-purpose registers (R0 through R11, AP, FP, PC, and SP); the per-process stack pointers (USP, SSP, KSP); the page table base and length registers (P0BR, P0LR, P1BR, and P1LR); and the processor status longword (PSL).

When SDA displays an address, it displays that address both in hexadecimal and as a symbol, if possible. If the address is within FFF_{16} of the value of a symbol, SDA displays the symbol plus the offset from the value of that symbol to the address. If more than one symbol's value is within FFF_{16} of the address, SDA displays the symbol whose value is the closest. If no symbols have values within FFF_{16} of the address, SDA displays no symbol. (For an example, see the description of the SHOW STACK command.)

8 Investigating System Failures

This section discusses how the operating system handles internal errors and suggests procedures that can aid you in determining the causes of these errors. To conclude, 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 the SDA Commands section.

8.1 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 such error.

To resolve the problem, you must find the reason for the bugcheck. Most 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 on CDROM.

Following are the steps you can take to diagnose an error:

1. Start the search for the error by locating the line of code that signaled the bugcheck. Invoke SDA and use the SHOW CRASH command to display the contents of the program counter (PC). The PC contains the address of the instruction immediately following the instruction that signaled the bugcheck.
2. Use the SHOW STACK command to display the contents of the stack. The PC often contains an address in the exception handler. This address is the address of the instruction that signaled the bugcheck, but not the address of the instruction that caused it. In this case, the address of the instruction that

caused the bugcheck is located on the stack. See Section 8.2 for information about how to proceed for several types of bugchecks.

3. Once you have found the address of the instruction that caused the bugcheck, you need to find the module in which the failing instruction resides. Use the SHOW DEVICE command to determine whether the instruction is part of a device driver.

   - If the module is not part of a driver, 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 programs.
   - If the module is not within a driver or other code supplied by Compaq, perform the following steps:

     a. Issue the following SDA command:

        ```
        SDA> SHOW EXECUTIVE
        ```

        This command shows the location and size of each of the loadable images that make up the executive.

     b. Compare the suspected address with the addresses of the system images.

     c. If the address is within one of the images, issue the following command:

        ```
        SDA> READ/EXECUTIVE  SYS$LOADABLE_IMAGES:
        ```

        This command loads the symbols that define locations within the loadable portion of the executive. (READ/EXECUTIVE is the default display.)

     d. Examine the failing address by issuing the following command:

        ```
        SDA> EXAMINE @PC
        ```

        SDA then displays the address in the PC as an offset from the nearest global symbol. This symbol might be the module's starting address, although it is possible that the code you are examining might not be in the module whose name is displayed.

4. To determine the general cause of the system failure, examine the code that signaled the bugcheck.

8.2 Fatal Bugcheck Conditions

Several conditions result in a bugcheck. Normally, these occasions are rare. When they do occur, it is likely that they are in the nature of a fatal exception or an illegal page fault occurring within privileged code. This section describes the symptoms of these bugchecks. A discussion of other exceptions and condition handling in general appears in the *OpenVMS System Services Reference Manual*.

8.2.1 Fatal Exceptions

An exception is fatal when it occurs while the following conditions exist:

- The process is using the interrupt stack.
- 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 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:

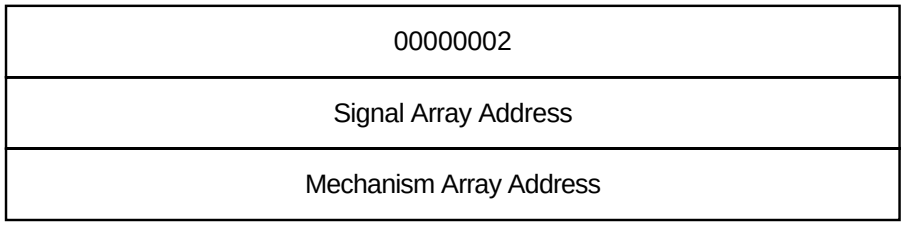
```
FATALEXCPT, Fatal executive or kernel mode exception

INVEXCEPTN, Exception while above ASTDEL or on interrupt stack

SSRVEXCEPT, Unexpected system service exception
```

Although several exception conditions are possible, access violations are the most common. When the hardware detects an access violation, information useful in finding the cause of the violation is pushed onto either the kernel stack or the interrupt stack. If the access violation occurs when the hardware is using the interrupt stack, this information appears on the interrupt stack.

The INVEXCEPTN, SSRVEXCEPT, and FATALEXCPT bugchecks place two argument lists, known as the mechanism and signal arrays, on the stack.

The SSRVEXCEPT and FATALEXCPT bugchecks push an additional argument list onto the stack above these arrays; INVEXCEPTN does not. This pointer array (see Figure SDA–1) contains the number 2 in its first longword, indicating that the following two longwords complete the array. The second longword contains the stack address of the **signal array**; the third contains the stack address of the **mechanism array**.

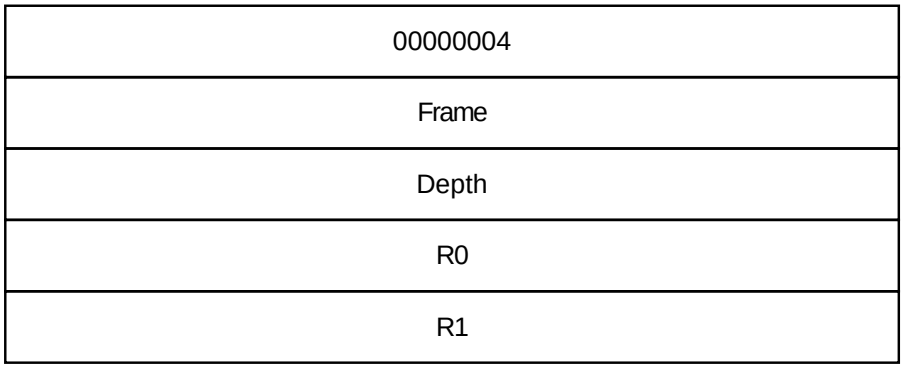
Figure SDA–1 Pointer Argument List on the Stack



ZK−1920−GE

The first longword of the **mechanism array** (see Figure SDA–2) contains a 4, indicating that the four subsequent longwords complete the array. These four longwords are used by the procedures that search for a condition handler and report exceptions.

Figure SDA–2 Mechanism Array



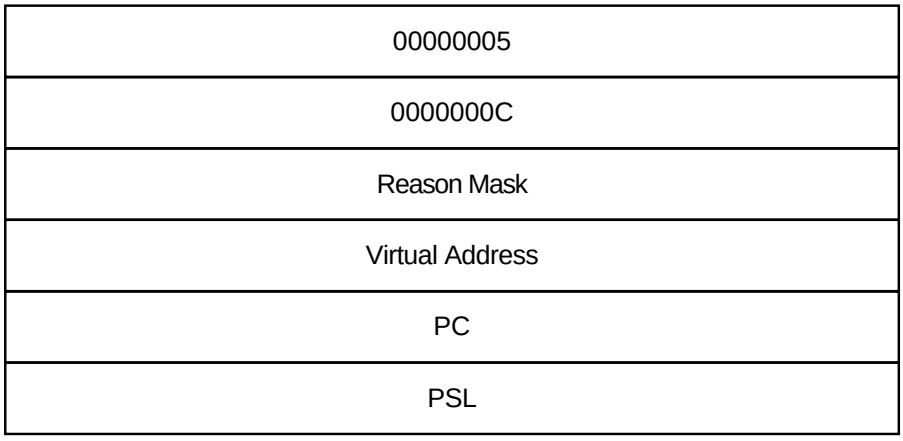
ZK−1921−GE

The values in the mechanism array are the following:

Value	Meaning
00000004	Number of longwords that follow. In a mechanism array, this value is always 4.
Frame	Address of the FP (frame pointer) of the establisher's call frame.
Depth	Depth of the search for a condition handler.
R0	Contents of R0 at the time of the exception.
R1	Contents of R1 at the time of the exception.

The **signal array** (see Figure SDA–3) appears somewhat further down the stack. A signal array contains the exception code, zero or more exception parameters, the PC, and the PSL. The size of a signal array can thus vary from exception to exception.

Figure SDA–3 Signal Array



ZK−1922−GE

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

Value	Meaning
00000005	Number of longwords that follow. For access violations, this value is always 5.
0000000C	Exception code. The value $0C_{16}$ represents an access violation. You can identify the exception code by using the SDA command EVALUATE/CONDITION.
Reason mask	Longword mask. If bit 0 of this 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.
Virtual address	Virtual address that the failing instruction tried to reference.
PC	PC whose execution resulted in the exception.
PSL	PSL at the time of the exception.

In the case of a fatal exception, you can find the code that signaled it by examining the PC in the signal array. Use the SHOW STACK command to display the stack in use when the failure occurred and then locate the mechanism and signal arrays. Once you obtain the PC, which points to the instruction that signaled the exception, you can identify the module where the instruction is located by following the instructions in Section 9.3.

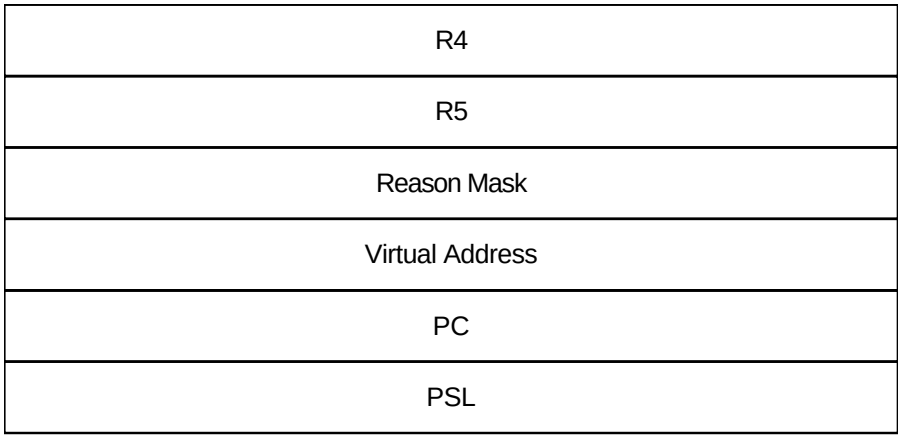
8.2.2 Illegal Page Faults

A PGFIPLHI bugcheck occurs when a page fault occurs while the interrupt priority level (IPL) is greater than 2 (IPL$_ASTDEL). When the system fails because of an illegal page fault, the following message appears on the console terminal:

```
PGFIPLHI, page fault with IPL too high
```

When an illegal page fault occurs, the stack appears as shown in Figure SDA–4.

Figure SDA–4 Stack Following an Illegal Page-Fault Error



ZK−1923−GE

Six longwords describe the error:

Longword	Contents
R4	Contents of R4 at the time of the bugcheck.
R5	Contents of R5 at the time of the bugcheck.
Reason mask	Longword mask. If bit 0 of this 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 an "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.
Virtual address	Virtual address being referenced by the instruction that caused the page fault.
PC	PC containing the address of the instruction that caused the page fault.
PSL	PSL at the time of the page fault.

If the operating system detects a page fault while the IPL is higher than IPL$_ ASTDEL, you can obtain the address of the instruction that caused the fault by examining the PC pushed onto the current operating stack. Follow the steps outlined in Section 9.3 to determine which module issued the instruction.

9 A Sample System Failure

This section steps through the analysis of a system failure using, as an example, a printer driver. Three events lead up to this failure:

1. The line printer goes off line for 3 hours.
2. The line printer comes back on line.
3. The operating system signals a bugcheck, writes information to the system dump file, and shuts itself down.

The following sections describe the actions to take in investigating the causes of this system crash.

9.1 Identifying the Bugcheck

First, invoke SDA to analyze the system dump file. The initialization message indicates the type of bugcheck that occurred as follows:

```
Dump taken on 31-JAN-1993 16:34:31.23
INVEXCEPTN, Exception while above ASTDEL or on interrupt stack

SDA>
```

An exception occurred that caused the system to signal a bugcheck, and signal and mechanism arrays have been created on the current operating stack.

9.2 Identifying the Exception

Use the SHOW STACK command to display the current operating stack. In this case, it is the interrupt stack. The following example shows the interrupt stack and the signal and mechanism arrays. See the SHOW STACK command for a complete description of the format of the stack display.

```
CPU 01 Processor stack
----------------------
Current operating stack (INTERRUPT)
        8006A378    8000844B    ACP$WRITEBLK+0A0
   .
   .
   .
  SP => 8006A398    7FFDC340
        8006A39C    8006A3A0
        8006A3A0    80004E7D    EXE$REFLECT+0D4
        8006A3A4    04080009
        8006A3A8    00000004
        8006A3AC    7FFDC368
        8006A3B0    FFFFFFFD
        8006A3B4    8001774E
        8006A3B8    0000074F
        8006A3BC    00000001
        8006A3C0    00000005
        8006A3C4    0000000C
        8006A3C8    00000000
        8006A3CC    80069E00
        8006A3D0    8005D003
        8006A3D4    04080000
        8006A3D8    80009604    EXE$FORKDSPTH+01C
   .
   .
   .
```

The mechanism array begins at address $8006A3A8_{16}$ and ends at address $8006A3B8_{16}$. Its first longword contains 00000004_{16}. The signal array begins at address $8006A3C0_{16}$ and ends at $8006A3D4_{16}$. Its first longword contains 00000005_{16} and its second longword contains $0000000C_{16}$. Examination of the signal array shows the following:

- The exception code is $0C_{16}$, indicating an access violation.
- The reason mask is zero, indicating that the instruction caused a protection violation (instead of a length violation) when it tried to read the location (rather than write to it).
- The virtual address that the instruction attempted to reference was $80069E00_{16}$.

- The PC of the instruction that referred to the bad virtual address was $8005D003_{16}$.

Issuing the SDA command EVALUATE/PSL 04080000 makes the following information apparent:

- The IPL was 8 at the time of the exception (shown by bits 16 through 20 of the PSL).
- The current operating stack was the interrupt stack (bit 26 of the PSL is set to 1).
- The process was executing in kernel mode at the time of the exception (shown by bits 24 and 25 of the PSL).

Use the SHOW PAGE_TABLE command to display the system page table, as shown in the following example. The page containing location $80069E00_{16}$ is not available to any access mode (a null page); thus, the virtual address is not valid.

```
SDA> SHOW PAGE_TABLE

System page table
-----------------

ADDRESS   SVAPTE   PTE       TYPE  PROT  BITS PAGTYP  LOC STATE TYPE REFCNT   BAK       SVAPTE  FLINK  BLINK
   .
   .
   .
80068400  80777B08 7C40FFC8  STX   UR       K
80068600  80777B0C 7C40FFC8  STX   UR       K
80068800  80777B10 7C40FFC8  STX   UR       K
80068A00  80777B14 7C40FFC8  STX   UR       K
80068C00  80777B18 7C40FFC8  STX   UR       K
80068E00  80777B1C 7C40FFC8  STX   UR       K
80069000  80777B20 7C40FFC8  STX   UR       K
80069200  80777B24 7C40FFC8  STX   UR       K
80069400  80777B28 7C40FFC8  STX   UR       K
80069600  80777B2C 7C40FFC8  STX   UR       K
80069800  80777B30 7C40FFC8  STX   UR       K
80069A00  80777B34 780016C9  TRANS UR       K SYSTEM FREELST 00   01    0   0040FFC8   80777B34  03AF  0E15
80069C00  80777B38 78000E15  TRANS UR       K SYSTEM FREELST 00   01    0   0040FFC8   80777B38  16C9  2592
-------- 40 NULL PAGES
   .
   .
   .
```

9.3 Locating the Source of the Exception

Because the printer went off line and then came back on line, as shown on the console listing in Section 9.2, the problem might exist in the driver code. SDA can help you determine which driver might contain the faulty code.

9.3.1 Finding the Driver by Using the Program Counter

The first step in determining whether the failing instruction is within a driver is to examine the PC in the signal array using the EXAMINE/INSTRUCTION command. This has two results:

- If possible, it displays the contents of the address as a MACRO instruction.
- It identifies the address as an offset from the symbol, *nn*DRIVER, if the address lies within the first FFF_{16} bytes of such a symbol. SDA defines a symbol in the form of *nn*DRIVER for each device driver connected to the system. This symbol represents the base of the driver prologue table (DPT). Each DPT is part of the device driver it describes and is immediately followed by that driver's code.

In the following example, the instruction that caused the exception is located within the printer driver.

```
SDA> EXAMINE/INSTRUCTION 8005D003
LPDRIVER+2B3   MOVB    (R3)+,(R0)
```

If SDA is unable to find a symbol within FFF_{16} bytes of the memory location you specify, it displays the location as an absolute address. This often, but not always, means the instruction that caused the exception is not part of a device driver.

To determine whether an instruction is part of a driver, use the SHOW DEVICE command to display the starting addresses and lengths of all the drivers in the system. If the address of the failing instruction falls within the range of addresses shown for a given driver, the failing instruction is a part of that driver. The following example shows a partial list of the drivers in the display generated by the SHOW DEVICE command.

```
I/O data structures
                          DDB list
                          --------

    Address    Controller     ACP       Driver      DPT   DPT size
    -------    ----------     ---       ------      ---   --------

    80000ECC    HELIUM$DBA    F11XQP    DBDRIVER   800F7AD0  08FD
    80001040    OPA                     OPERATOR   80001622  0061
    8000126C    MBA                     MBDRIVER   800015B0  0578
    80001460    NLA                     NLDRIVER   800015E9  05A3
    801E2800    HELIUM$DMA    F11XQP    DMDRIVER   800B5CB0  0AA0
    801E2980    HELIUM$DLA    F11XQP    DLDRIVER   800B6A50  08D0
   .
   .
   .
```

9.3.2 Calculating the Offset into the Driver's Program Section

The offsets that SDA displays from *nn*DRIVER are actually offsets from the DPT. As such, these offsets do not exactly correspond to the offsets shown in driver listings, which represent offsets from the beginning of the program section (PSECT) in which a given instruction appears. Because a driver usually contains more than one PSECT, you must use the driver's map to determine the location of the failing instruction within the driver listing.

To calculate the location of the instruction within the driver listing, refer to the "Program Section Synopsis" section of the driver's map. Determine in which PSECT the offset given by SDA occurs and subtract the base of the PSECT from the offset. You can then use the resulting figure as an index into the driver listing.

If SDA does not display the address as an offset from *nn*DRIVER, but the address *is* within the address range of a driver in the SHOW DEVICE display, you must first subtract the address of the DPT from the failing address. Using the result as the offset, you can then follow the steps previously outlined for determining the index of the instruction into a driver listing.

9.4 Finding the Problem Within the Routine

To find the problem within the routine, examine the printer's driver code. In the system failure discussed in this example, the instruction that caused the exception is MOVB (R3)+,(R0). To check the contents of R3, use the EXAMINE command as follows:

```
SDA> EXAMINE R3

R3: 80069E00 "...."
```

The invalid virtual address, as recorded in the signal array, is stored in R3. In the following driver code excerpt, the instruction in question appears at line 599. It is likely that the contents of R3 have been incremented too many times.

```
581 STARTIO:
582      MOVL    UCB$L_IRP(R5),R3     ;Retrieve address of I/O packet
583      MOVW    IRP$L_MEDIA+2(R3),-
584              UCB$W_BOFF(R5)       ;Set number of characters to print
585      MOVL    UCB$L_SVAPTE(R5),R3  ;Get address of system buffer
586      MOVAB   12(R3),R3            ;Get address of data area
587      MOVL    UCB$L_CRB(R5),R4     ;Get address of CRB
588      MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4 ;Get device CSR address
589 ;
590 ; START NEXT OUTPUT SEQUENCE
591 ;
592
593 10$: ADDL3   #LP_DBR,R4,R0        ;Calculate address of data buffer register
594      MOVZWL  UCB$W_BOFF(R5),R1    ;Get number of characters remaining
595      MOVW    #^X8080,R2           ;Get control register test mask
596      BRB     25$                  ;Start output
597 20$: BITW    R2,(R4) ❶            ;Printer ready or have paper problem?
598      BLEQ    30$                  ;If LEQ not ready or paper problem
599      MOVB    (R3)+,(R0) ❷         ;Output next character
600      ASHL    #1,G^EXE$GL_UBDELAY,-(SP)    ;Delay 3*2 u-seconds
601 24$: SOBGEQ  (SP),24$             ;Delay loop calibrated to machine speed
602      ADDL    #4,SP                ;Pop extra longword off stack
603 25$: SOBGEQ  R1,20$ ❸             ;Any more characters to output?
604      BRW     70$                  ;All done, BRW to set return status
```

Explanations of the circled numbers in the example are in Section 9.4.1.

9.4.1 Examining the Routine

The MOVB instruction is part of a routine that reads characters from a buffer and writes them to the printer. The routine contains the loop of instructions that starts at the label 20$ and ends at 25$. This loop executes once for each character in the buffer, performing these steps:

❶ The driver checks the printer's status register to see if the printer is ready.

❷ If the printer is ready, the driver gets a character from the buffer and moves it to the printer's data register, to which R0 points.

❸ It then decrements R1, which contains the count of characters left to print. If R1 contains a number greater than 0, control is passed back to the instruction at 20$, and the loop begins again.

Steps 1 and 2 are repeated until the contents of R1 are 0 or the printer signals that it is not ready.

If the printer signals that it is not ready, the driver transfers control to 30$ (line 598), the beginning of a routine that waits for an interrupt from the printer. When the printer becomes ready, it interrupts the driver and execution of the loop resumes.

Examine the code to determine which variables control the loop.

The byte count (BCNT) is the number of characters in the buffer. Note that BCNT is set by a function decision table (FDT) routine and that this routine sets the value of BCNT to the number of characters in the buffer. In line 586, the starting address of a buffer that is BCNT bytes in size is moved into R3.

Note also that the number of characters left to be printed is represented by the byte offset (BOFF), the offset into the buffer at which the driver finds the next character to be printed. This value controls the number of times the loop is executed.

Because the exception is an access violation, either R3 or R0 must contain an incorrect value. You can determine that R0 is probably valid by the following logic:

- The instruction at 10$ (ADDL3 #LP_DBR,R4,R0) places an address in R0 and R0 is not modified again until the failing instruction (line 599).
- The value in R4 at the time that the instruction at 10$ is executed was derived from the addresses of the device's unit control block (UCB) (line 587) and CRB (line 599). Although it is possible that these data structures might contain wrong information, it is unlikely.

Thus, the contents of R3 seem to be the cause of the failure.

The most likely reason that the contents of R3 are wrong is that the MOVB instruction at line 599 executes too many times. You can check this by comparing the contents of UCB$W_BOFF and UCB$W_BCNT. If UCB$W_BOFF contains a larger value than that in UCB$W_BCNT, then R3 contains a value that is too large, indicating that the MOVB instruction has incremented the contents of R3 too many times.

9.4.2 Checking the Values of Key Variables

Because the start-I/O routine requires that R5 contain the address of the printer's UCB, and because several other instructions reference R5 without error before any instruction in the loop does, you can assume that R5 contains the address of the right UCB. To compare BOFF and BCNT, use the command FORMAT @R5 to display the contents of the UCB, as shown in the following session.

```
SDA> READ SYS$SYSTEM:SYSDEF.STB
SDA> FORMAT @R5

8005D160    UCB$L_FQFL      800039A8
            UCB$L_RQFL
            UCB$W_MB_SEED
            UCB$W_UNIT_SEED
8005D164    UCB$L_FQBL      800039A8
            UCB$L_RQBL
8005D168    UCB$W_SIZE          0122
8005D16A    UCB$B_TYPE        10
8005D16B    UCB$B_FIPL      34
            UCB$B_FLCK
```

```
   .
   .
   .
8005D1C8    UCB$L_SVAPTE    80062720
8005D1CC    UCB$W_BOFF          0795
8005D1CE    UCB$W_BCNT      006D
8005D1D0    UCB$B_ERTCNT          00
8005D1D1    UCB$B_ERTMAX        00
8005D1D2    UCB$W_ERRCNT    0000
   .
   .
   .
SDA>
```

If you have only one printer in your system configuration, you do not need to use the FORMAT command. Instead, you can use the command SHOW DEVICE LP. Because only one printer is connected to the processor, only one UCB is associated with a printer for SDA to display.

The output produced by the FORMAT @R5 command shows that UCB$W_BOFF contains a value greater than that in UCB$W_BCNT; it should be smaller. Therefore, the value stored in BOFF is incorrect.

Thus, the value of BOFF is not the number of characters that remain in the buffer. This value is used in calculating an address that is referenced at an elevated IPL. When this address is within a null page (unreadable in all access modes), an attempt to reference it causes the system to fail.

9.4.3 Identifying and Correcting the Defective Code

Examine the printer driver code to locate all instructions that modify UCB$W_ BOFF. The value changes in two circumstances:

- Immediately after the driver detects that the printer is not ready and that the problem is not a paper problem (line 609).
- When the wait-for-interrupt routine's timeout count of 12 seconds is exhausted (lines 616 and 630). At this time, the contents of R1, plus 1, are stored in UCB$W_BOFF (line 631).

When the printer times out, the driver should not modify UCB$W_BOFF. It does so, however, in line 631. The driver should modify the contents of UCB$W_BOFF only when it is certain that the printer printed the character. When the printer times out, this is not the case. Furthermore, the wait-for-interrupt routine preserves only registers R3, R4, and R5, so that only those registers can be used unmodified after the execution of the wait-for-interrupt routine. Thus, the use of R1 in line 631 is an error.

To correct the problem, change the WFIKPCH argument (line 616) so that, when the printer times out, the WFIKPCH macro transfers control to 50$ rather than to 40$.

```
607
608 30$: BNEQ    40$                   ;If NEQ paper problem
609     ADDW3   #1,R1,UCB$W_BOFF(R5) ;Save number of characters remaining
610     DEVICELOCK -
611             LOCKADDR=UCB$L_DLCK(R5),-  ;Lock device interrupts
612             SAVIPL=-(SP)          ;Save current IPL
613     BITW    #^X80,LP_CSR(R4)      ;Is it ready now?
614     BNEQ    35$                   ;If NEQ, yes, it's ready
615     BISB    #^X40,LP_CSR(R4)      ;Set interrupt enable
616     WFIKPCH 40$,#12               ;Wait for ready interrupt
617     IOFORK                        ;Create a fork process
618     BRB     10$                   ;  ...and start next output
619
620 35$:
621     DEVICEUNLOCK -
622             LOCKADDR=UCB$L_DLCK(R5),-  ;Unlock device interrupts
623             NEWIPL=(SP)+          ;Restore IPL
624     CLRW    LP_CSR(R4)            ;Disable device interrupts
625     BRB     10$                   ;Go transfer more characters
626 ;
627 ; PRINTER HAS PAPER PROBLEM
628 ;
629
630 40$: CLRL    UCB$L_LP_OFLCNT(R5)  ;Clear offline counter
631     ADDW3   #1,R1,UCB$W_BOFF(R5) ;Save number of characters remaining
632 50$: CLRW    LP_CSR(R4)            ;Disable printer interrupt
633     IOFORK                        ;Lower to fork level
634     BBS     #UCB$V_CANCEL,UCB$W_STS(R5),80$  ;If set, cancel I/O operation
635     TSTW    LP_CSR(R4)            ;Printer still have paper problem?
636     BLSS    55$                   ;If LSS yes
637     MOVL    #15,UCB$L_LP_TIMEOUT(R5)  ;Set timeout value
638     BRB     10$                   ; ...and start next output
```

10 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 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 (if it is "hung"), see the examples of causing system failures in Section 10.2.

If you are generating a system crash in response to a system hang, be sure to record the PC at the time of the system halt as well as the contents of the general registers. Submit this information to Compaq, along with the Software Performance Report (SPR) and a copy of the generated system dump file.

10.1 Meeting Crash Dump Requirements

The following requirements must be met before the 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$SYSTEM: 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 1.1) or, if the DUMPSTYLE system parameter is set, large enough to accommodate a subset dump (see Section 1.1.2).

  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).

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

10.2 Examples of How to Cause System Failures

The following examples show the sequence of console commands needed to cause a system failure on each type of processor. In each instance, after halting the processor and examining its registers, you place the equivalent of –1 (for example, $FFFFFFFF_{16}$) into the PC. The value placed in the PSL sets the processor access mode to kernel and the IPL to 31. After these commands are executed, an INVEXCEPTN bugcheck is reported on the console terminal, followed by a listing of the contents of the processor registers.

The console volume of most processors contains a command file named either CRASH.COM or CRASH.CMD, which you can execute to perform these commands. Note that the console sessions recorded in this section omit much of the information the console displays in response to the listed commands.

VAX 85*x*0/8700/88*x*0

The following series of console commands causes a system failure on the VAX 85*x*0/8700/88*x*0 systems. (Note that the console prompt for the VAX 8810, 8820, and 8830 systems is PS-CIO-0> and not >>>.)

```
$ Ctrl/P
>>> SET CPU CURRENT_PRIMARY
>>> HALT
?00       Left CPU -- CPU halted
          PC = 8001911C
>>> @CRASH
!
! Command procedure to force bugcheck via access violation
!
SET VERIFY
SET CPU CURRENT_PRIMARY    !Select primary
EXAMINE PSL                !Display PSL
        M 00000000 00420008
EXAMINE/I/NEXT 4 0
.
.
.

DEPOSIT PC FFFFFFFF        !Set PC=-1 to force ACCVIO
DEPOSIT PSL 41F0000        !Set IPL=31, interrupt stack
CONTINUE                   !Execute from PC=-1
```

VAX 82*x*0/83*x*0, VAXstation 3520/3540, 6000 Series, and 9000 Series

The following console commands cause a system failure on a VAX 82*x*0/83*x*0 system, a VAXstation 3520/3540 system, a VAX 6000 series system, or a VAX 9000 series system.

```
$ Ctrl/P
        PC = 80008B1F
>>> E P
>>> E/I 0
>>> E/I +
>>> E/I +
>>> E/I +
>>> E/I +
>>> D/G F FFFFFFFF
>>> D P 41F0000
>>> C
```

VAX 8600/8650

The following console commands cause a system failure on the VAX 8600/8650 systems.

```
$ Ctrl/P
>>> @CRASH
    SET QUIET OFF          !Make clearer
    SET ABORT OFF          !Don't abort on E/VIR command
    HALT
        CPU stopped, INVOKED BY CONSOLE (CSM code 11)
        PC 80008B1F
    UNJAM                  !Clear the way
    E PSL                  !Display PSL
        U PSL 00000000
    E/I/N:4 0              !Display stack pointers
   .
   .
   .
    E SP                   !Get current stack pointers
        G 0E 80000C40
    E/vir/next:40 @        !Dump top of stack
   .
   .
   .
    D PC FFFFFFFF          !Invalidate the PC
    D PSL 1F0000           !Kernel mode, IPL 31
    SET ABORT ON           !Restore abort flag
    SET QUIET ON           !Shut output off
    CONTINUE               !Force a machine check
```

VAX-11/780 and VAX-11/785

The following console commands cause a system failure on the VAX-11/780 and VAX-11/785 processors.

```
$ Ctrl/P
>>> @CRASH
HALT                       !Halt system, examine PC,
HALTED AT 80008A89

EXAMINE PSL                !PSL,
00000000

EXAMINE/INTERN/NEXT:4 0    !and all stack pointers

DEPOSIT PC = -1            !Invalidate PC
DEPOSIT PSL = 41F0000       !Kernel mode, IPL 31
```

```
CONTINUE
```

VAX-11/750

The following code causes a system failure on a VAX-11/750. On this processor, the HALT command is a NOP; a Ctrl/P automatically halts the processor.

```
$ Ctrl/P
>>> H
>>> E P
>>> E/I 0
>>> E/I +
>>> E/I +
>>> E/I +
>>> E/I +
>>> D/G F FFFFFFFF
>>> D P 41F0000
>>> C
```

MicroVAX 3400/3600/3900 Series, VAXstation/MicroVAX 3100, VAXstation/MicroVAX 2000, MicroVAX II, and VAX 4000 Series

To force a crash of a MicroVAX, you must first halt the processor. (After you halt the processor, press the HALT button again so that it is popped out and is not illuminated.) Then, issue the following console commands:

```
>>> E PSL
>>> E/I/N:4 0
>>> D PC FFFFFFFF
>>> D PSL 41F0000
>>> C
```

VAX-11/730

The following console commands cause a system failure on a VAX-11/730. Ctrl/P automatically halts the processor.

```
$ Ctrl/P
>>> H
>>> E PSL
>>> E/I/N:4 0
>>> D PC FFFFFFFF
>>> D PSL 1F0000
>>> C
```

SDA Usage Summary

The System Dump Analyzer is a utility that you can use to help determine the causes of system failures. This utility is also useful for examining the running system.

Format

analyze {/CRASH_DUMP [/RELEASE] filespec | /SYSTEM}
[/SYMBOL=system-symbol-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.

Usage Summary

The following table summarizes how to perform key SDA operations.

Operation	Command	Explanation or Requirements
Invoke SDA to analyze a system dump	$ ANALYZE/CRASH_ DUMP *filename*	If you do not specify a file name, SDA prompts you for one. Reading the dump file usually requires system privilege (SYSPRV), but your system manager can allow less privileged processes to read 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	$ ANALYZE/SYSTEM	Your process must have change-mode-to-kernel (CMKRNL) privilege. You cannot specify a file name with the /SYSTEM qualifier.
Send all output from SDA to a file	SDA> SET OUTPUT *filename*	The file produced is 132 columns wide and is formatted for output to a printer.
Redirect the output to your terminal	$ SET OUTPUT SYS$OUTPUT	
Send a copy of all the commands you enter and all the output those commands produce to a file	SDA> SET LOG *filename*	The file produced is 132 columns wide and is formatted for output to a printer.
Exit an SDA display or the SDA utility	SDA> EXIT	If SDA is in display mode, you must use the EXIT command twice: once to exit display mode and a second time to exit SDA.

SDA 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. Table SDA–10 briefly describes the SDA qualifiers.

Table SDA–10 Descriptions of SDA Qualifiers

Qualifier	Description
/CRASH_DUMP	Invokes SDA to analyze a specified dump file
/RELEASE	Invokes SDA to release those blocks that are occupied by a crash dump in a specified system paging file
/SYMBOL	Specifies a system symbol table for SDA to use in place of the system symbol table it uses by default (SYS$SYSTEM:SYS.STB)
/SYSTEM	Invokes SDA to analyze a running 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 Section 2 for additional information on crash dump analysis.

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 SYS$SYSTEM:PAGEFILE.SYS
   ```

   This command invokes SDA to analyze a crash dump stored in the system paging file.

/RELEASE

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

Format

/RELEASE filespec

Parameter

filespec
Name of the system page file (SYS$SYSTEM:PAGEFILE.SYS). 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

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

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

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

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

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

Example

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

This command invokes SDA to release to the paging file those blocks in SYS$SYSTEM:PAGEFILE.SYS occupied by a crash dump.

/SYMBOL

Specifies a system symbol table for SDA to use in place of the system symbol table it uses by default (SYS$SYSTEM:SYS.STB).

Format

/SYMBOL =system-symbol-table

Parameter

system-symbol table
File specification of the SDA system symbol table needed to define symbols required by SDA to analyze a dump from a particular 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 SYS$SYSTEM:SYS.STB by default. When you *do* specify the /SYMBOL qualifier, SDA assumes the default disk and directory to be *SYS$DISK*: that is, the disk and directory specified in your last SET DEFAULT command. If SDA is given a file that is not a system symbol table in the /SYMBOL qualifier, it halts with a fatal error.

Description

The /SYMBOL qualifier allows you to specify a system symbol table, other than SYS$SYSTEM:SYS.STB, to load into the SDA symbol table. This might be necessary, for instance, to analyze a crash dump taken on a processor running a different version of OpenVMS.

You can use the /SYMBOL qualifier whether you are analyzing a system dump or a running system.

Example

```
$ ANALYZE/CRASH_DUMP/SYMBOL=SYS$CRASH:SYS.STB SYS$SYSTEM
```

This command invokes SDA to analyze the crash dump stored in SYS$SYSTEM:SYSDUMP.DMP, using the system symbol table at SYS$CRASH:SYS.STB.

/SYSTEM

Invokes SDA to analyze a running system.

Format

/SYSTEM

Parameters

None.

Description

See Section 3 for a full discussion of using SDA to analyze a running system.

You cannot specify the /CRASH_DUMP 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.

SDA Commands

Table SDA–11 briefly describes the SDA commands that are explained fully in the following section.

Table SDA–11 Descriptions of SDA Commands

Command	Description
@ (Execute Procedure)	Causes SDA to execute SDA commands contained in a file
ATTACH	Switches control of your terminal from your current process to another process in your job
COPY	Copies the contents of the dump file to another file
DEFINE	Assigns a value to a symbol or associates an SDA command with a terminal key
EVALUATE	Computes and displays the value of the specified expression in both hexadecimal and decimal
EXAMINE	Displays either the contents of a location or range of locations in physical memory, or the contents of a register
EXIT	Exits from an SDA display or exits from the SDA utility
FORMAT	Displays a formatted list of the contents of a block of memory
HELP	Displays information about the SDA utility, its operation, and the format of its commands
READ	Loads the global symbols contained in the specified object module into the SDA symbol table
REPEAT	Repeats execution of the last command issued
SEARCH	Scans a range of memory locations for all occurrences of a specified value
SET CPU	Selects a processor to become the SDA current CPU
SET LOG	Initiates or discontinues the recording of an SDA session in a text file
SET OUTPUT	Redirects output from SDA to the specified file or device
SET PROCESS	Selects a process to become the SDA current process
SET RMS	Changes the options shown by the SHOW PROCESS/RMS command
SHOW CALL_ FRAME	Displays the locations and contents of the longwords representing a procedure call frame
SHOW CLUSTER	Displays connection manager and system communications services (SCS) information for all nodes in a cluster
SHOW CONNECTIONS	Displays information about all active connections between SCS processes or a single connection
SHOW CPU	Displays information about the state of a processor at the time of the system failure

(continued on next page)

Table SDA–11 (Cont.) Descriptions of SDA Commands

Command	Description
SHOW CRASH	In the analysis of a system failure, displays information about the state of the system at the time of the failure; in the analysis of a running system, provides information identifying the system
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
SHOW EXECUTIVE	Displays the location and size of each loadable image that makes up the executive
SHOW HEADER	Displays the header of the dump file
SHOW LAN	Displays information contained in various local area network (LAN) data structures
SHOW LOCK	Displays information about all lock management locks in the system, cached locks, or a specified lock
SHOW LOGS	Displays information about transaction logs currently open for the node
SHOW PAGE_ TABLE	Displays a range of system page table entries, the entire system page table, or the entire global page table
SHOW PFN_ DATA	Displays information that is contained in the page lists and PFN database
SHOW POOL	Displays information about the disposition of paged and nonpaged memory, nonpaged dynamic storage pool, and paged dynamic storage pool
SHOW PORTS	Displays those portions of the port descriptor table (PDT) that are port independent
SHOW PROCESS	Displays the software and hardware context of any process in the balance set
SHOW RESOURCE	Displays information about all resources in the system or about a resource associated with a specific lock
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
SHOW RSPID	Displays information about response IDs (RSPIDs) of all SCS connections or, optionally, a specific SCS connection
SHOW SPINLOCKS	Displays information taken from the data structures that provide system synchronization in a multiprocessing environment
SHOW STACK	Displays the location and contents of the four process stacks (of the SDA current process) and the interrupt stack (of the SDA current CPU)
SHOW SUMMARY	Displays a list of all active processes and the values of the parameters used in swapping and scheduling those processes

(continued on next page)

Table SDA–11 (Cont.) Descriptions of SDA Commands

Command	Description
SHOW SYMBOL	Displays the hexadecimal value of a symbol and, if the value is equal to an address location, the contents of that location
SHOW TRANSACTIONS	Displays information about all transactions on the node or about a specified transaction
SPAWN	Creates a subprocess of the process currently running SDA, copying the context of the current process to the subprocess
VALIDATE QUEUE	Validates the integrity of the specified queue by checking the pointers in the queue

@ (Execute Procedure)

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 Procedure 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 to the file information about the crash and its context, a description of the process executing at the time of the process, the contents of the stack on which the crash occurred, and a list of the processes active on the CPU that crashed.

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

You cannot nest command procedures.

ATTACH

Switches control of your terminal from your current process to another process in your job.

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 parent process of the current 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 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 SET DEFAULT command. You must supply at least the file name.

Description

Each time the system fails, it copies the contents of physical memory and the hardware context of the current process (as directed by the DUMPSTYLE parameter) into the file SYS$SYSTEM:SYSDUMP.DMP (or the paging file), overwriting its current contents. If you do not save this crash dump elsewhere, it will be overwritten the next time 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 (from within SYS$MANAGER:SYSTARTUP_VMS.COM) to execute the COPY command. This ensures that a copy of the dump file is made each time the system fails.

The COPY command does not affect the contents of SYS$SYSTEM:SYSDUMP.DMP.

If you are using the paging file (SYS$SYSTEM:PAGEFILE.SYS) as the dump file instead of SYSDUMP.DMP, you can use the COPY command to explicitly release the blocks of the paging file that contain the dump, thus making them available for paging. Although the copy operation succeeds nonetheless, the release operation requires that your process have change-mode-to-kernel (CMKRNL) privilege. Once the dump pages have been released from the paging file, the dump information in those pages might be lost. You need to analyze the copy of the dump created by the COPY command.

Example

```
SDA> COPY SYS$CRASH:SAVEDUMP
```

The COPY command copies the dump file into the file SYS$CRASH:SAVEDUMP.DMP.

DEFINE

Assigns a value to a symbol or associates an SDA command with a terminal key.

Format

DEFINE [symbols-name [=] expression | /KEY key-name command |[/qualifier....]]

Parameters

symbol-name
Name, containing from 1 to 31 alphanumeric characters, that identifies the symbol. See Section 7.2.4 for a description of SDA symbol syntax and a list of default symbols.

expression
Definition of the symbol's value. See Section 7.2 for a discussion of the components of SDA expressions.

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

Key Name	Key Designation
PF1	LK201, VT100, VT52 Red
PF2	LK201, VT100, VT52 Blue
PF3	LK201, VT100, VT52 Black
PF4	LK201, VT100
KP0 . . . KP9	Keypad 0–9
PERIOD	Keypad period
COMMA	Keypad comma
MINUS	Keypad minus
ENTER	Keypad Enter
UP	Up arrow
DOWN	Down arrow
LEFT	Left arrow
RIGHT	Right arrow
E1	LK201 Find
E2	LK201 Insert Here
E3	LK201 Remove
E4	LK201 Select
E5	LK201 Prev Screen
E6	LK201 Next Screen
HELP	LK201 Help
DO	LK201 Do
F7 . . . F20	LK201 function keys

command
SDA command the key is to be defined as. The command must be enclosed in quotation marks (" ").

Qualifiers

/ECHO
/NOECHO
Determines whether the equivalence string is displayed on the terminal screen after the defined key has been pressed. The /NOECHO qualifier functions only with the /TERMINATE qualifier. The default is /ECHO.

/IF_STATE=(state-name, . . .)
/NOIF_STATE
Specifies a list of one or more states, one of which must be in effect for the key definition to be in effect. States are placed in effect by the /SET_STATE qualifier, which is described in this section.

The **state-name** is an alphanumeric string, enclosed in quotation marks (" "). By including several state names, you can define a key to have the same function in all the specified states. If you specify only one state name, you can omit the parentheses.

If you omit the /IF_STATE qualifier—or use /NOIF_STATE—the current state is used.

/KEY
Defines a key as an SDA command. You need only to press the defined key and the Return key to issue the command. If you use the /TERMINATE qualifier as well, you do not need to press the Return key.

When you define some keys as SDA commands, you must press Ctrl/V first before those keys will execute the commands. This is because of the escape sequences the keys generate and the way the terminal driver handles those escape sequences. The following keys, when defined as SDA commands, must be preceded by a Ctrl/V:

Key Name	Key Designation
LEFT	Left arrow
RIGHT	Right arrow
F7 . . . F14	LK201 function keys

/SET_STATE=state-name
Causes the key being defined to cause a key state change rather than issue an SDA command. When you use the /SET_STATE qualifier, you supply the name of a key state in place of the **key-name** parameter. In addition, you must define the **command** parameter as a pair of quotation marks (" ").

The key state can be any name you think appropriate. For example, you can define the PF1 key to set the state to GOLD and use the /IF_STATE=GOLD qualifier to allow two definitions for other keys, one in the GOLD state and one in the non-GOLD state.

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

The DEFINE/KEY command associates an SDA command with the specified key, in accordance with any specified qualifiers.

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 SDA.

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 that the EXAMINE command can inspect.

2.
```
SDA>  DEFINE NEXT = @PC
SDA>  EXAMINE/INSTRUCTION NEXT
NEXT:   MOVL @00(R6),R0
```

Symbol NEXT defines the address contained in the program counter so that you can use the symbol in an EXAMINE/INSTRUCTION command.

3.
```
SDA>  DEFINE VEC SCH$GL_PCBVEC
SDA>  EXAMINE VEC
VEC: 80B7D31C  ".O.."
```

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

4.
```
SDA>  DEFINE COUNT = 7
SDA>  DEFINE RESULT = COUNT * COUNT
SDA>  EVALUATE RESULT
Hex = 00000031   Decimal = 49       PR$_SBIS
                                                RESULT
```

The first DEFINE command assigns the value 7 to symbol COUNT. The second DEFINE command defines RESULT to be the result of the evaluation of an arithmetic expression using the symbol COUNT. Evaluation of RESULT shows that system symbol PR$_SBIS has an equivalent value.

5.
```
SDA> DEFINE/KEY PF1 "SHOW STACK"
SDA> PF1 SHOW STACK RETURN
Process stacks (on CPU 00)
------------------------
Current operating stack (KERNEL):

              7FFE8DD4  00001703     SGN$C_MAXPGFL+703
              7FFE8DD8  80127920
              7FFE8DDC  00000000
              7FFE8DE0  00000000
              7FFE8DE4  00000000
              7FFE8DE8  00000000
              7FFE8DEC  7FF743E4
              7FFE8DF0  7FF743CC

       SP =>  7FFE8DF4  8000E646     EXE$CMODEXEC+1EE
              7FFE8DF8  7FFEDE96     SYS$CMKRNL+006
              7FFE8DFC  03C00000
```

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.

6.
```
SDA> DEFINE/KEY/TERMINATE PF1 "SHOW STACK"
SDA> PF1 SHOW STACK
Process stacks (on CPU 00)
------------------------
Current operating stack (KERNEL):
   .
   .
   .
```

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

7.
```
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 command defines PF1 as a key that sets command state GREEN. The trailing pair of quotation marks is required syntax, indicating that no command is to be executed when you press this key.

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.

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|/PSL|/PTE|/SYMBOLS} expression

Parameter

expression
SDA expression to be evaluated. Section 7.2 describes the components of SDA expressions.

Qualifiers

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

/PSL
Evaluates the specified expression in the format of a processor status longword.

/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 that are 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 several such symbols.

Description

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 if the offset is small enough for the selected symbol to be considered useful.

Examples

1.
```
SDA> EVALUATE -1
Hex = FFFFFFFF   Decimal = -1           PR$_XSID_N8NNN
```

   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 = 00000001   Decimal = 1            ACP$V_SWAPGRP
                                        ACP$V_WRITECHK
                                        EVT$_EVENT
```

   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 finite number of symbols are displayed by default.

3.
```
SDA> DEFINE TEN = A
SDA> EVALUATE TEN
Hex = 0000000A   Decimal = 10           EXE$V_FATAL_BUG
                                        SGN$C_MINWSCNT
                                        TEN
```

   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 = 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.

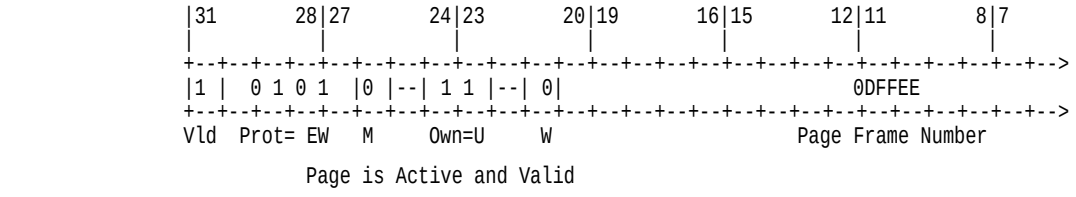
5.
```
SDA> EVALUATE/CONDITION 80000018
%SYSTEM-W-EXQUOTA, exceeded quota
```

   This example shows the output of an EVALUATE/CONDITION command.

6.
```
SDA> EVALUATE/PSL 04080009
        CMP TP FPD IS CURMOD PRVMOD IPL DV FU IV T N Z V C
         0   0  0   1  KERN   KERN   08  0  0  0 0 1 0 0 1
```

   SDA interprets the entered value 04080009 as though it were a processor status longword (PSL) and displays the resulting field values of that longword.

7.
```
SDA> EVALUATE/PTE ABCDFFEE
```



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

EXAMINE

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

Format

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

Parameter

location

Location in memory to be examined. You can represent a location by any valid SDA expression (see Section 7.2). To examine a range of locations, use the following format:

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 either of the following:

- The process that was executing at the time the system failed (if you are examining a crash dump)
- 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 4. Use of the /INSTRUCTION qualifier increases the default address as appropriate to the translation of the instruction. 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 parts of the writable system region, displaying the contents of memory in hexadecimal longwords. Do not specify parameters when you use this qualifier.

/CONDITION_VALUE

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

/INSTRUCTION

Translates the contents of the specified range of memory locations into MACRO instruction format. If more than 16 bytes are specified in the range, /INSTRUCTION processing might skip some bytes at the beginning of the range to ensure that SDA is properly synchronized with the start of each instruction. You can override this synchronization by specifying the /NOSKIP qualifier.

The length of the instruction displayed varies according to the opcode and addressing mode. If SDA cannot decode a memory location, it issues the following message:

```
%SDA-E-NOINSTRAN, cannot translate instruction
```

When you use this qualifier with the EXAMINE command, SDA calculates subsequent default addresses by adding the length of the last instruction, including all operands, to the last address examined.

/NOSKIP
Causes the EXAMINE command not to skip any bytes in the range when translating the contents of memory into MACRO instructions. The /NOSKIP qualifier causes the execution of the /INSTRUCTION qualifier by default.

/NOSUPPRESS
Inhibits the suppression of zeros when displaying memory with one of the following qualifiers: /ALL, /P0, /P1, /SYSTEM.

/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.

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

/PTE
Interprets and displays the specified longword 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.

/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:  80002119  ".!.."
```

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 : 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 columns of information:

- Each of the first four columns represents a longword of memory, the contents of which are displayed in hexadecimal format.
- The fifth column lists the ASCII value of each byte in each longword displayed in the previous four columns.
- The sixth 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. For example:

```
SDA> EXAMINE 100:220

Virtual locations 00000100 through 000001FF are not in physical memory

0130011A 0120011B 0130011E 0110011F  ......0... ...0.     00000200
01200107 02300510 04310216 04210218  ..!...1...0... .     00000210
01100103 01100104 01200105 01200106  .. ... .........     00000220
```

Addresses 100_{16} through $1FF_{16}$ do not exist in memory, as the message indicates. SDA displays the contents of those addresses that do exist (200_{16} through 220_{16}).

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

```
Zeros suppressed from 'loc1' to 'loc2'
```

Note that if you make a mistake specifying a virtual address for the EXAMINE command and you are examining global page table entries, your system may crash with a bugcheck. This occurs rarely and only when you use ANALYZE/SYSTEM.

Decoding Locations

You can translate the contents of memory locations into MACRO 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.

If the specified range of locations does not begin on an instruction boundary, SDA skips bytes until it locates the next valid instruction and issues the following message:

```
%SDA-W-INSKIPPED, unreasonable instruction stream - n bytes skipped
```

In this message, *n* represents the number of bytes that SDA could not translate.

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/SYSTEM

System Region Memory
--------------------
00040039 8FBC0010 00040038 8FBC0010  ....8.......9...     800000000
   .
   .
   .
```

   This example shows only the first two lines of the display generated by the EXAMINE/SYSTEM command. Note that in the dump the fifth byte from the right contains the value 38_{16}. The ASCII value of 38_{16}, the character 8, is represented in the fifth character from the left in column 5.

   Likewise, the thirteenth byte from the right in the dump columns contains the value 39_{16}. The ASCII value of 39_{16} is 9, and 9 is represented in the ASCII column as the thirteenth character from the left.

2.
```
SDA> EXAMINE/PSL G1268

        CMP TP FPD IS CURMOD PRVMOD IPL DV FU IV T N Z V C
         1   0  0   0  KERN   KERN   00  0  1  0 1 1 1 0 0
```

   This example shows the display produced by the EXAMINE/PSL command. The address of the longword examined is 80001268_{16}.

3.
```
SDA> EXAMINE/PTE G775F480


  |31       28|27       24|23       20|19       16|15       12|11        8|7
  |           |           |           |           |           |           |
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-->
  |1 |  1 1 1 0  |1 |--| 0 0 |--| 0|                          00F0F4
  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-->
  Vld  Prot= URKW M     Own=K     W                     Page Frame Number

            Page is Active and Valid
```

   The EXAMINE/PTE command displays and formats the system page table entry at $8775F480_{16}$.

4.
```
SDA> EXAMINE/TIME EXE$GQ_SYSTIME

18-FEB-1993 02:07:25.88
```

   The EXAMINE/TIME command displays the formatted value of the system time quadword (EXE$GQ_SYSTIME).

EXIT

Exits from an SDA display or from the SDA utility.

Format

EXIT

Parameters

None.

Qualifiers

None.

Description

If SDA is displaying information about a video display terminal—and if that information extends beyond one screen—SDA displays a **screen overflow prompt** at the bottom of the screen:[4]

```
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 screen overflow prompt does not immediately precede the SDA> prompt, entering EXIT causes your process to cease executing the SDA utility. When you issue EXIT within a command procedure (either the SDA initialization file or a command procedure invoked with the @ command), SDA terminates execution of the procedure and returns to the SDA prompt.

[4] On hardcopy terminals, SDA does not display such a prompt.

FORMAT

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

Format

FORMAT [/qualifier] location

Parameter

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

Qualifier

/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 of a data block, as described in the Description section. The **block-type** can be the symbolic prefix of any data structure.

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

Normally, you use the FORMAT command without the /TYPE qualifier. Used in this manner, it examines the byte in the structure that contains the type of the structure. In most data structures, this byte occurs at an offset of $0A_{16}$ into the structure. If this byte does not contain a valid block type, the FORMAT command halts with this message:

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

However, if this byte does contain a valid block type, SDA checks the next byte (offset $0B_{16}$) for a secondary block type. When SDA has determined the type of block, it searches for the symbols that correspond to that type of block.

If SDA cannot find the symbols associated with the block type it has found (or that you specified in the /TYPE qualifier), it issues this message:

```
No "block-type" symbols found to format this block
```

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

```
$ READ SYS$SYSTEM:SYSDEF.STB
```

Certain data structures do *not* contain a block type at offset $0A_{16}$. If this byte contains information other than a block type—or the byte does not contain a valid block type—SDA 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.

Example

```
SDA> READ SYS$SYSTEM:SYSDEF.STB
SDA> FORMAT 800B81F0

800B81F0   UCB$L_FQFL              80000F10
           UCB$L_RQFL
           UCB$W_MB_SEED
           UCB$W_UNIT_SEED
800B81F4   UCB$L_FQBL              800026A8
           UCB$L_RQBL
800B81F8   UCB$W_SIZE                  00E0
800B81FA   UCB$B_TYPE                10
800B81FB   UCB$B_FLCK              07
800B81FC   UCB$L_ASTQFL            800F80E0
           UCB$L_FPC
           UCB$T_PARTNER
800B8200   UCB$L_ASTQBL            8002CF80
           UCB$L_FR3
800B8204   UCB$L_FIRST             8002CA00
           UCB$L_FR4
           UCB$W_MSGMAX
           UCB$W_MSGCNT
   .
   .
   .
```

From SYS$SYSTEM:SYSDEF.STB, the READ command loads into SDA's symbol table the symbols needed for formatting system data structures. The FORMAT command displays the data structure that begins at $800B81F0_{16}$, 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 $800B8204_{16}$ has three designations: UCB$L_FIRST and UCB$L_FR4, alternative names for the longword; and the two subfields, UCB$W_MSGMAX and UCB$W_MSGCNT.

The contents of each field appear to the right of the symbolic name of the field. Thus, the contents of UCB$L_FIRST are $8002CA00_{16}$.

HELP

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

Format

HELP [command-name]

Parameter

command-name
Command for which you need information.

You can also specify the following keywords in place of **command-name**.

Keyword	Function
CPU_CONTEXT	Describes the concept of CPU context as it governs the behavior of SDA in uniprocessor and multiprocessor environments
EXECUTE_ COMMAND	Causes SDA to execute SDA commands contained in a file
EXPRESSIONS	Prints a description of SDA expressions
INITIALIZATION	Describes the circumstances under which SDA executes an initialization file when first invoked
OPERATION	Describes how to operate SDA at your terminal and by means of the site-specific startup procedure
PROCESS_CONTEXT	Describes the concept of process context as it governs the behavior of SDA in uniprocessor and multiprocessor environments
SYMBOLS	Consists of up to 31 letters and numbers, and can include the dollar sign ($) and underscore (_) characters. When you invoke SDA, it reads in the global symbols from symbols table psect of SYS$BASE_ IMAGE.EXE, and from REQSYSDEF.STB, a required subset of the symbols in the file SYSDEF.STB. You can add other symbols to SDA's symbol table by using the DEFINE and READ commands.

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 those commands and topics for which you can request help, as follows:

```
  Information available:

  ATTACH     COPY       CPU_Context     DEFINE          EVALUATE   EXAMINE
  Execute_Command       EXIT       Expressions          FORMAT     HELP
  Initialization        Operation  Process_Context      READ       REPEAT
  SEARCH     SET        SHOW       SPAWN      Symbols   VALIDATE QUEUE

Topic?
```

READ

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

Format

READ {/EXECUTIVE *directory-spec* | [RELOCATE=*expression*] | *filespec*}

Parameter

filespec
Name of the device, directory, and file that contains the object module from which you want to copy global symbols. The **filespec** defaults to SYS$DISK:[*default-dir*]*filename*.STB, where SYS$DISK and *[default-dir]* represent the disk and directory specified in your last SET DEFAULT command. You must specify a file name.

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. (See Table SDA–13 for a list of those images.)

The **directory-spec** is the name of the directory containing the loadable images of the executive. This parameter defaults to SYS$LOADABLE_IMAGES.

/RELOCATE=expression
Adds the value of **expression** to the value of each symbol in the symbol table file to be read. You can use the /RELOCATE qualifier only if you also specify a **filespec**. The /RELOCATE qualifier is useful for examining images that are position independent and are loaded at a base of zero.

Description

The READ command symbolically identifies locations in memory for which the default symbol table (SYS$SYSTEM:SYS.STB) provides no definition. In other words, the required global symbols are located in modules that have been compiled and linked separately from the executive.[5]

The object module file specified in the READ command can be one of the following:

- Output of a compiler or assembler (for example, an .OBJ file)
- Output generated by the linker qualifier /SYMBOL_TABLE (for example, an .STB file)

Most often the object module file is a file provided by the operating system in SYS$SYSTEM or SYS$LOADABLE_IMAGES. Many SDA applications, for instance, need to load the definitions of system data structures by issuing a READ command specifying SYS$SYSTEM:SYSDEF.STB. Others require the definitions of specific global entry points within the executive image that are contained within those object modules included in the executive.

[5] SDA extracts no local symbols from the object module.

Table SDA–12 lists those object module files provided in SYS$SYSTEM. Table SDA–13 lists those loadable images in SYS$LOADABLE_IMAGES that define locations within the executive image.

Table SDA–12 Modules Containing Global Symbols and Data Structures Used by SDA

File	Contents
CLUSTRLOA.STB	Symbols for loadable VAXcluster management code
DCLDEF.STB	Symbols for the DCL interpreter
IMGDEF.STB	Symbols for the image activator
NETDEF.STB	Symbols for DECnet data structures
RMSDEF.STB	Symbols that define RMS internal and user data structures and RMS$_*xxx* completion codes
SCSDEF.STB	Symbols that define data structures for system communications services
SYSDEF.STB	Symbols that define system data structures, including the I/O database
TCPIP$NET_GLOBALS.STB[1]	Data structure definitions for TCP/IP internet driver, execlet, and ACP data structures
TCPIP$NFS_GLOBALS.STB[1]	Data structure definitions for TCP/IP NFS server
TCPIP$PROXY_GLOBALS.STB[1]	Data structure definitions for TCP/IP proxy execlet
TCPIP$PWIP_GLOBALS.STB[1]	Data structure definitions for TCP/IP PWIP driver, and ACP data structures
TCPIP$TN_GLOBALS.STB[1]	Data structure definitions for TCP/IP TELNET/RLOGIN server driver data structures

[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 SDA–13 Modules Defining Global Locations Within the Executive Image

File	Contents
CPULOA.EXE	Processor-specific data and initialization routines
ERRORLOG.EXE	Error logging routines and system services
EVENT_FLAGS_AND_ASTS.EXE	Event flag and AST delivery routines and system services
EXCEPTION.EXE	Bugcheck and exception handling routines and those system services that declare condition and exit handlers
IMAGE_MANAGEMENT.EXE	Image activator and the related system services

(continued on next page)

Table SDA–13 (Cont.) Modules Defining Global Locations Within the Executive Image

File	Contents
IO_ROUTINES.EXE	$QIO system service, related system services (for example, SYS$CANCEL and SYS$ASSIGN), and supporting routines
LMF$GROUP_TABLE.EXE	Data for valid, licensed product groups
LOCKING.EXE	Lock management routines and system services
LOGICAL_NAMES.EXE	Logical name routines and system services
MESSAGE_ROUTINES.EXE	System message routines and system services (including SYS$SNDJBC and SYS$GETTIM)
PAGE_MANAGEMENT.EXE	System pager, its supporting routines, and page management system services (including SYS$CRMPSC, SYS$CREDEL, and SYS$ADJSTK)
PRIMITIVE_IO.EXE	Console I/O routines
PROCESS_MANAGEMENT.EXE	Scheduler, report system event, and supporting routines and system services
RECOVERY_UNIT_SERVICES.EXE	Recovery unit system services
RMS.EXE	Global symbols and entry points for RMS
SECURITY.EXE	Security management routines and system services
SYSDEVICE.EXE	Mailbox driver and null driver
SYSGETSYI.EXE	Get System Information system service (SYS$GETSYI)
SYSLICENSE.EXE	Licensing system service (SYS$LICENSE)
SYSMSG.EXE	System messages
SYSTEM_PRIMITIVES.EXE	Miscellaneous basic system routines, including those that allocate system memory, maintain system time, create fork processes, and control mutex acquisition
SYSTEM_SYNCHRONIZATION.EXE	Routines that enforce synchronization in a multiprocessing system
TCPIP$BGDRIVER.STB[1]	TCP/IP internet driver
TCPIP$INETACP.STB[1]	TCP/IP internet ACP
TCPIP$INTERNET_SERVICES.STB[1]	TCP/IP internet execlet

[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.

(continued on next page)

Table SDA–13 (Cont.) Modules Defining Global Locations Within the Executive Image

File	Contents
TCPIP$NFS_SERVICES.STB[1]	Symbols for the TCP/IP NFS server
TCPIP$PROXY_SERVICES.STB[1]	Symbols for the TCP/IP proxy execlet
TCPIP$PWIPACP.STB[1]	TCP/IP PWIP ACP
TCPIP$PWIPDRIVER.STB[1]	TCP/IP PWIP driver
TCPIP$TNDRIVER.STB[1]	TCP/IP TELNET/RLOGIN server driver
WORKING_SET_ MANAGEMENT.EXE	Swapper, its supporting routines, and working set management system services

[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.

Examples

1.
```
SDA>  READ SYS$SYSTEM:SYSDEF.STB
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYSEXE]SYSDEF.STB;1
```

The READ command causes SDA to add all the global symbols in SYS$SYSTEM: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>  EXAM/INST EXE$QIO+2;4

EXE$QIO+00002:  CHMK    #001F
EXE$QIO+00006:  RET
SDA>  EXAM/INST V_EXE$QIO

%SDA-E-BADSYM, unknown symbol "V_EXE$QIO"
SDA>  READ/RELOCATE=IO_ROUTINES SYS$LOADABLE_IMAGES:IO_ROUTINES.EXE

%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]IO_ROUTINES.EXE;1
SDA>  EXAM/INST EXE$QIO+2;4

EXE$QIO+00002:  MOVZBL  04(AP),R3
EXE$QIO+00006:  CMPB    R3,#3F
SDA>  EXAM/INST V_EXE$QIO+2;4

V_EXE$QIO+00002:  CHMK    #001F
V_EXE$QIO+00006:  RET
```

This SDA session shows that the initial examination of the instructions at EXE$QIO+2 and EXE$QIO+6 produces the vector for the system service, not the system service code itself. The subsequent READ instruction brings into the SDA symbol table the global symbols defined for the system's I/O routines, including one that redefines the entry point of the system service to be the start of the routine EXE$QIO. Thus, the second examination of the same memory locations produces the first two instructions in the routine. The READ command creates a special symbol, V_EXE$QIO, that points to the system service vector.

3.
```
SDA>  SHOW STACK

Process stacks (on CPU 01)
--------------------------
Current operating stack (KERNEL):

                7FF8F2B0  806BA870
                7FF8F2B4  7FF8F4C0
                7FF8F2B8  8016F33E     PAGE_MANAGEMENT+0053E
   .
   .
   .
SDA>  READ/RELOCATE=PAGE_MANAGEMENT SYS$LOADABLE_IMAGES:PAGE_MANAGEMENT.EXE

%SDA-I-READSYM, reading symbol table SYS$COMMON:[SYS$LDR]PAGE_MANAGEMENT.EXE;1

SDA>  SHOW STACK

Process stacks (on CPU 01)
--------------------------
Current operating stack (KERNEL):

                7FF8F2B0  806BA870
                7FF8F2B4  7FF8F4C0
                7FF8F2B8  8016F33E     MMG$LOCK_SYSTEM_PAGES+00188
   .
   .
   .
```

The initial SHOW STACK command contains an address that SDA resolves into an offset from the PAGE_MANAGEMENT module of the executive. The READ command loads the corresponding symbols into the SDA symbol table such that the reissue of the SHOW STACK command subsequently identifies the same location as an offset within a specific page management routine.

4.
```
SDA>  READ/EXEC

%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]RECOVERY_UNIT_SERVICES.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]RMS.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]CPULOA.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]LMF$GROUP_TABLE.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]SYSLICENSE.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]SYSGETSYI.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]SYSDEVICE.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]MESSAGE_ROUTINES.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]EXCEPTION.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]LOGICAL_NAMES.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]SECURITY.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]LOCKING.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]PAGE_MANAGEMENT.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]WORKING_SET_MANAGEMENT.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]IMAGE_MANAGEMENT.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]EVENT_FLAGS_AND_ASTS.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]IO_ROUTINES.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]PROCESS_MANAGEMENT.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]ERRORLOG.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]PRIMITIVE_IO.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]SYSTEM_SYNCHRONIZATION.EXE;1
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]SYSTEM_PRIMITIVES.EXE;1
```

This READ command brings all global symbols defined in the modules of SYS$SYSTEM:SYS.EXE (as listed in Table SDA–13) into the SDA symbol table. Included in its results is the work performed by the READ commands illustrated in the two previous examples. The READ/EXECUTIVE command, however, does not load those symbols contained in the modules described in Table SDA–12.

REPEAT

Repeats execution of the last command issued. On terminal devices, the KP0 key performs the same function as the REPEAT command.

Format

REPEAT

Parameters

None.

Qualifiers

None.

Description

The REPEAT command is useful for stepping through a linked list of data structures or for examining a sequence of memory locations.

Examples

```
1.  SDA>  FORMAT @IOC$GL_DEVLIST
    8000B540   DDB$L_LINK              8000B898
    8000B544   DDB$L_UCB               8000B5E0
    8000B548   DDB$W_SIZE                  0044
       .
       .
       .
    8000B554   DDB$B_NAME_LEN                03
               DDB$T_NAME              "OPA"
       .
       .
       .
    SDA>  FORMAT @.
    8000B898   DDB$L_LINK              8000BBE0
    8000B89C   DDB$L_UCB               8000B9E0
    8000B8A0   DDB$W_SIZE                  0044
       .
       .
       .
    8000B8AC   DDB$B_NAME_LEN                03
               DDB$T_NAME              "MBA"
    SDA>  KP0
    8000BBE0   DDB$L_LINK              807F85C0
    8000BBE4   DDB$L_UCB               8000BC80
    8000BBE8   DDB$W_SIZE                  0044
       .
       .
       .                                     )
    8000BBF4   DDB$B_NAME_LEN                03
               DDB$T_NAME              "NLA"
```

This series of FORMAT commands pursues the chain of device data blocks (DDBs) from the system global symbol IOC$GL_DEVLIST. The second FORMAT command is constructed so that it refers to the contents of the address at the current location (see Section 7.2.4 for a discussion of SDA symbols). Subsequently,

pressing the KP0 key—or issuing the REPEAT command—is sufficient to display each DDB in the device list.

2.
```
SDA> SHOW CALL_FRAME

Call Frame Information
----------------------
        Call Frame Generated by CALLG Instruction

Condition Handler       7FFE7D78  00000000
SP Align Bits = 00      7FFE7D7C  00000000
   Saved AP             7FFE7D80  7FFE7DC0      CTL$GL_KSTKBAS+005C0
   Saved FP             7FFE7D84  7FFE7D94      CTL$GL_KSTKBAS+00594

   .
   .
   .
SDA> SHOW CALL_FRAME/NEXT_FP

Call Frame Information
----------------------
        Call Frame Generated by CALLS Instruction

Condition Handler       7FFE7D94  00000000
SP Align Bits = 00      7FFE7D98  20FC0000
   Saved AP             7FFE7D9C  7FFED024      CTL$GL_KSTKBAS+005E4
   Saved FP             7FFE7DA0  7FFE7DE4      SYSTEM_PRIMITIVES+020AA
   .
   .
   .
SDA> REPEAT

Call Frame Information
----------------------
        Call Frame Generated by CALLG Instruction

Condition Handler       7FFE7DE4  00000000
   .
   .
   .
```

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 (see Section 7.2). To search a range of locations, use the following format:

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
Indication of 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 7.2.

Qualifiers

/LENGTH={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.

/STEPS={QUADWORD|LONGWORD|WORD|BYTE}
Specifies the granularity 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 to determine the next location to undergo the comparison. If you do not specify the /STEPS qualifier, the SEARCH command uses a step factor of one longword.

Description

SEARCH displays each location as each value is found.

Examples

1.
```
SDA> SEARCH GB81F0;500 60068
Searching from 800B81F0 to 800B86F0 in LONGWORD steps for 00060068...
Match at 800B8210
SDA>
```

The SEARCH command finds the value 0060068 in the longword at 800B8210.

2.
```
SDA> SEARCH/STEPS=BYTE 80000000;1000 6
Searching from 80000000 to 80001000 in BYTE steps for 00000006...
Match at 80000A99
SDA>
```

   The SEARCH command finds the value 00000006 in the longword at 80000A99.

3.
```
SDA> SEARCH/LENGTH=WORD 80000000;2000 6
Searching from 80000000 to 80002000 in LONGWORD steps for 0006...
Match at 80000054
Match at 800001EC
Match at 800012AC
Match at 800012B8
SDA>
```

   The SEARCH command finds the value 0006 in the longword locations 80000054, 800001EC, 800012AC, and 800012B8.

SET CPU

Selects a processor to become the SDA current CPU.

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 SDA current CPU context defaults to that of the processor that caused the system to fail. When analyzing a crash from a multiprocessing system, you might find it useful at times to examine the context of another processor in the configuration.

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

SET CPU **cpu-id**
SHOW CPU **cpu-id**
SHOW CRASH

The following commands also change SDA CPU context if the **name** or index number (**nn**) refers to a current process:

SET PROCESS **name**
SET PROCESS/INDEX=**nn**
SHOW PROCESS **name**
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, SDA process context is undefined and no process-specific information is available until you set SDA process context to that of a specific process.

See Section 4 for further discussion on the way in which SDA maintains its context information.

You cannot use the SET CPU command when examining the running system with SDA.

Example

```
$ ANALYZE/CRASH SYS$SYSTEM:SYSDUMP.DMP

Dump taken on 22-FEB-1993 14:22:17.66
NOBUFPCKT, Required buffer packet not present

SDA> SHOW CPU

CPU 01 Processor crash information
---------------------------------

CPU 01 reason for Bugcheck: NOBUFPCKT, Required buffer packet not present
   .
   .
   .
SDA> SHOW STACK

CPU 01 Processor stack
----------------------
Current operating stack (INTERRUPT):

        80DAFB4C    8018BC20
        80DAFB50    7FFC653E
   .
   .
   .
SDA> SET CPU 00
SDA> SHOW CPU

CPU 00 Processor crash information
---------------------------------

CPU 00 reason for Bugcheck: CPUEXIT, Shutdown requested by another CPU
   .
   .
   .
SDA> SHOW STACK

CPU 00 Processor stack
----------------------
Current operating stack (INTERRUPT):

        8016ABD8    00011F4C
        8016ABDC    00010F56
   .
   .
   .
SDA> SHOW CRASH

System crash information
------------------------
Time of system crash: 22-FEB-1993 14:22:17.66
   .
   .
   .
SDA> SHOW STACK

CPU 01 Processor stack
----------------------
Current operating stack (INTERRUPT):
```

```
80DAFB4C    8018BC20
80DAFB50    7FFC653E
.
.
.
```

The series of SHOW CPU and SHOW STACK commands in this example illustrates the switching of CPU context within an SDA session:

1. When you first invoke SDA, it is, by default, within the CPU context of the processor that caused the crash (CPU 01). This is illustrated by the first set of SHOW CPU and SHOW STACK commands.
2. The SET CPU 00 command explicitly changes SDA CPU context to that of CPU 00, as illustrated by the second sequence of SHOW CPU and SHOW STACK commands.

   Note that a SHOW CPU 00 command would have the same effect as the two commands SET CPU 00 and SHOW CPU, changing the SDA CPU context in addition to displaying the processor-specific information. Unlike the SHOW CPU **cpu-id** command, no display is associated with the SET CPU **cpu-id** command.
3. The SHOW CRASH command resets the SDA CPU context to that of the processor that caused the crash (CPU 01).

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 SET DEFAULT command. 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.

There are the following differences 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 such that 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 all 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 LOG command does not record SDA error messages in its log file. The SET OUTPUT command can record SDA error messages in its listing file.
- The SET OUTPUT command generates 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 have used the SET OUTPUT command to redirect output to a listing file, you cannot use a SET LOG command to direct the same output to a log file.

SET OUTPUT

Redirects output from SDA to the specified file or device.

Format

SET OUTPUT 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 SET DEFAULT command. You must specify a file name.

Description

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

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
```

If you use the SET OUTPUT command to send the SDA output to a listing file, 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.

SET PROCESS

Selects a process to become the SDA current process.

Format

SET PROCESS {process-name|/INDEX=nn|/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 ($) character, and the underscore (_) character can also be included in the string. If you include characters other than these, you must enclose the entire string in quotation marks (" ").

Qualifiers

/INDEX=nn
Specifies the process to be made current by its index into the system's list of software process control blocks (PCBs). You can supply either of the following values for **nn**:

- The process index itself
- The process identification (PID) or extended PID longword, from which SDA extracts the correct index

To obtain these values for any given process, issue the SDA command SHOW SUMMARY.

/SYSTEM
Specifies that the system process be made the SDA current process. Each system (uniprocessor or multiprocessor) uses a single system process control block (PCB) and process header (PHD) as dummy structures, located in system space, that record the system working set, global section table, global page table, and other systemwide data.

Description

When you issue an SDA command such as an EXAMINE command, 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, its process context defaults to that of the process that was current at the time of the crash. If the crash occurred on a multiprocessing system, SDA sets the CPU context to that of the processor that crashed the system and the process context 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 **name** or /INDEX=**nn**. The process specified by this command becomes the current process for SDA until you exit SDA or change SDA process context by issuing one of the following commands:

SET PROCESS/INDEX=**nn**
SET PROCESS **process-name**
SHOW PROCESS/INDEX=**nn**

In the analysis of a crash dump from a multiprocessing system, changing process context can involve 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 will automatically change its CPU context to that of the CPU on which that process is current. The following commands can have this effect if **process-name** or index number (**nn**) refers to a current process:

SET PROCESS **process-name**
SET PROCESS/INDEX=**nn**
SHOW PROCESS **process-name**
SHOW PROCESS/INDEX=**nn**

See Section 4 for further discussion on the way in which SDA maintains its context information.

Example

```
SDA> SHOW PROCESS
Process index: 0012   Name: NETACP   Extended PID: 28C00092
------------------------------------------------------------
Process status:  00149001   RES,WAKEPEN,NOACNT,PHDRES,LOGIN

PCB address              800F1140    JIB address              801FDA00
PHD address              80477200    Swapfile disk address    01000F01
   .
   .
   .

SDA> SHOW SUMMARY
Current process summary
-----------------------

 Extended Indx Process name    Username    State Pri   PCB      PHD    Wkset
-- PID -- ---- -------------- ----------- ----- --- -------- -------- -----
 28C00080 0000 SWINGER                     COM    0 80002100 80001F88     0
 28C00081 0001 SWAPPER                     HIB   16 800023C8 80002250     0
 28C00483 0003 KLINGON         KLINGON     MWAIT  6 8010FEA0 803F8600   323
 28C00085 0005 ERRFMT          SYSTEM      COM   10 800B5A10 8061DA00    69
 28C00087 0007 OPCOM           SYSTEM      LEF    7 800C7000 80227A00    71
   .
   .
   .
SDA>SET PROCESS ERRFMT
SDA> SHOW PROCESS
Process index: 0005   Name: ERRFMT   Extended PID: 28C00085
--------------
Process status:  00040001   RES,PHDRES

PCB address              800B5A10    JIB address              801E5C00
   .
   .
   .
```

The first SHOW PROCESS command shows the current process to be NETACP. The SHOW SUMMARY command shows the names of the processes that exist.

The SET PROCESS command sets the current process to ERRFMT, as shown by the second SHOW PROCESS command. Note that the SET PROCESS command could also have been issued as one of the following:

```
SDA> SET PROCESS/INDEX=5
SDA> SET PROCESS/INDEX=801E5C00
```

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 SDA–14 lists those keywords that you can use as options.

Table SDA–14 SET RMS Command Keywords for Displaying Process RMS Information

Keyword	Meaning
[NO]ALL[:**ifi**][1]	All control blocks (default)
[NO]ASB	Asynchronous context block
[NO]BDB	Buffer descriptor block
[NO]BDBSUM	BDB summary page
[NO]BLB	Buffer lock block
[NO]BLBSUM	Buffer lock summary page
[NO]CCB	Channel control block
[NO]DRC	Directory cache
[NO]FAB	File access block
[NO]FCB	File control block
[NO]FWA	File work area
[NO]GBD	Global buffer descriptor
[NO]GBDSUM	GBD summary page
[NO]GBH	Global buffer header
[NO]GBSB	Global buffer synchronization block
[NO]IDX	Index descriptor
[NO]IFAB[:**ifi**][1]	Internal FAB
[NO]IFB[:**ifi**][1]	Internal FAB
[NO]IRAB	Internal RAB
[NO]IRB	Internal RAB
[NO]JFB	Journaling file block
[NO]NAM	Name block
[NO]NWA	Network work area

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

(continued on next page)

Table SDA–14 (Cont.) SET RMS Command Keywords for Displaying Process RMS Information

Keyword	Meaning
[NO]PIO	Image I/O (NOPIO), the default, or process I/O (PIO)[2]
[NO]RAB	Record access block
[NO]RLB	Record lock block
[NO]RU	Recovery unit structures, including the recovery unit block (RUB), recovery unit stream block (RUSB), and recovery unit file block (RUFB)
[NO]SFSB	Shared file synchronization block
[NO]WCB	Window control block
[NO]XAB	Extended attribute block
[NO]*	Current list of options displayed by the SHOW RMS command

[2] Specifying the PIO option causes the SHOW PROCESS/RMS command to display the indicated structures for process-permanent file I/O.

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

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 an indication of the information provided by various displays.) You can examine the options that are currently selected by issuing a SHOW RMS 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.

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 BLB, BLBSUM, and RLB to the list of data structures that the SHOW PROCESS/RMS command currently displays.

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 **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.

SHOW CALL_FRAME

Displays the locations and contents of the longwords 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 longword contained in 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 to use the /NEXT_FP qualifier to the command.

Description

Whenever a procedure is called using CALLG or CALLS instructions, information is stored on the stack of the calling routine in the form of a procedure call frame. Figure SDA–5 illustrates the format of a call frame.[6]

The SHOW CALL_FRAME command interprets the contents of the designated call frame and displays whether the call frame was generated by a CALLG or CALLS instruction. If it locates nonzero bits in the portion of the second longword that represents the upper byte of the processor status word (PSW), it presents a message that indicates the fault or trap in effect. For example:

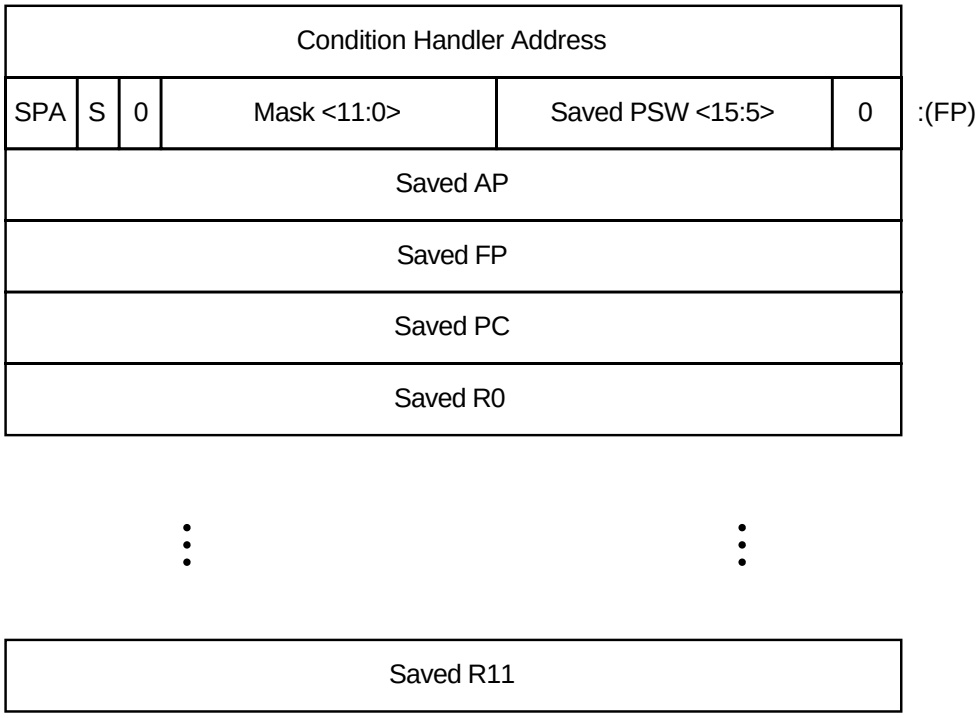
```
Nonzero PSW Bits (15:8) => Reserved Operand Fault on RET
```

SHOW_CALL_FRAME then produces four columns of information:

- The components of the call frame.
- The virtual addresses that are part of the call frame.
- The contents of the longwords at these addresses.
- A symbolic representation of the contents of each longword, if possible. SDA does not attempt to symbolize the second longword in the call frame (mask-PSW longword), which contains the register save mask and the processor status word (PSW).

[6] In Figure SDA–5, the second longword contains the stack pointer alignment (SPA) bits, which indicate the zero to three bytes needed to align the frame to a longword boundary. The S bit is set if the frame resulted from a CALLS instruction; it is clear if it resulted from a CALLG instruction.

Figure SDA–5 Call Frame



ZK−6564−GE

The SHOW CALL_FRAME command follows this listing with an indication of how many bytes were used to align the call frame to a longword boundary.

For call frames generated by a CALLS instruction, the SHOW CALL_FRAME instruction displays the argument list to the call frame in three columns containing the virtual address of each item, its contents, and its symbolic representation.

All valid procedure call frames begin on a longword boundary. If the specified address expression does not begin on a longword boundary, the call frame is invalid and SDA displays the following message:

```
Invalid Call Frame:   Start Address Not On Longword Boundary
```

If you attempt to format an address that is not a call frame or is an invalid call frame (that is, bit 28 of the second longword is not 0), SDA displays the following message:

```
Invalid Call Frame:   Bit 28 is Set in "Mask-PSW" Longword
```

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 : 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
----------------------
        Call Frame Generated by CALLG Instruction

Condition Handler       7FFE7D78  00000000
SP Align Bits = 00      7FFE7D7C  00000000
   Saved  AP            7FFE7D80  7FFE7DC0      CTL$GL_KSTKBAS+005C0
   Saved  FP            7FFE7D84  7FFE7D94      CTL$GL_KSTKBAS+00594
   Return PC            7FFE7D88  8015303F      EXCEPTION+0043F
Align Stack by 0 Bytes =>

SDA> SHOW CALL_FRAME/NEXT_FP

Call Frame Information
----------------------
        Call Frame Generated by CALLS Instruction

Condition Handler       7FFE7D94  00000000
SP Align Bits = 00      7FFE7D98  20FC0000
   Saved  AP            7FFE7D9C  7FFED024
   Saved  FP            7FFE7DA0  7FFE7DE4      CTL$GL_KSTKBAS+005E4
   Return PC            7FFE7DA4  801D58AA      MMG$IMGRESET+00066
        R2              7FFE7DA8  7FFE7DD0      CTL$GL_KSTKBAS+005D0
        R3              7FFE7DAC  7FFDB9F8
        R4              7FFE7DB0  8026C720
        R5              7FFE7DB4  7FFDBA00
        R6              7FFE7DB8  7FFE6300      CTL$A_DISPVEC+00500
        R7              7FFE7DBC  00000003
Align Stack by 0 Bytes =>
Argument List           7FFE7DC0  00000003
                        7FFE7DC4  7FFE7DD0      CTL$GL_KSTKBAS+005D0
                        7FFE7DC8  00000000
                        7FFE7DCC  00000000

SDA> SHOW CALL_FRAME/NEXT_FP

Call Frame Information
----------------------
        Call Frame Generated by CALLG Instruction

Condition Handler       7FFE7DE4  00000000
SP Align Bits = 00      7FFE7DE8  00000000
   Saved  AP            7FFE7DEC  7FFED024
   Saved  FP            7FFE7DF0  7FFECFF8
   Return PC            7FFE7DF4  8015303F      EXCEPTION+0043F
Align Stack by 0 Bytes =>
```

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 {/CSID=csid|/NODE=name|/SCS}

Parameters

None.

Qualifiers

/CSID=csid
Displays VAXcluster system information for a specific VAXcluster member node. The value **csid** is the cluster system identification number (CSID) of the node to be displayed.[7]

/NODE=name
Displays VAXcluster system information for a specific VAXcluster member node. The value **name** is the name of the node to be displayed.

/SCS
Displays a view of the cluster as seen by SCS.

Description

By default, the SHOW CLUSTER command provides a view of the VAXcluster system from the perspective of the connection manager. When you use the /SCS qualifier, however, SHOW CLUSTER provides a view of the cluster from the perspective of the port driver or drivers.

VAXcluster as Seen by the Connection Manager
The SHOW CLUSTER command provides a series of displays.

The **VAXcluster 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 a quorum is present

The **CSB list** displays information about the VAXcluster system blocks (CSB) currently in operation; there is one CSB assigned to each node of the cluster. For each CSB, the **CSB list** displays the following information:

- Its address
- Name of the VAXcluster node it describes
- CSID associated with the node

[7] 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.

- Number of votes (if any) provided by the node
- Its state[8]
- Its status

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 SDA is being run.

The **cluster failover control block** display provides detailed information concerning the cluster failover control block (CLUFCB), and 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 Show Cluster utility section of the *OpenVMS System Management Utilities Reference Manual* for information about the flags for VAXcluster nodes.)

VAXcluster 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 VAXcluster member, HSC, UDA, 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
VAXcluster data structures
--------------------------
                  --- VAXcluster Summary ---
        Quorum   Votes   Quorum Disk Votes   Status Summary
        ------   -----   -----------------   --------------
          2       3            1            quorum
```

[8] For information about the state and status of nodes, see the description of the ADD command in the Show Cluster utility section of the *OpenVMS System Management Utilities Reference Manual*.

```
                        --- CSB list ---
Address   Node    CSID      Votes   State     Status
-------   ----    ----      -----   -----     ------

803686F0  SOLLY   000100C8    1     open    member,qf_active
80368550  GUS     000100C9    1     open    member,qf_active
80367B90  DORIS   000100C5    1     open    member,qf_active

                --- Cluster Block (CLUB) 801C3F70 ---
Flags: 10080001 cluster,init,quorum

Quorum/Votes                 2/3    Last transaction code          02
Quorum Disk Votes              1    Last trans. number           1126
Nodes                          3    Last coordinator CSID    00000000
Quorum Disk             $255$DUA2   Last time stamp       26-MAR-1993
Found Node SYSID     0000000008A0                            18:52:32
Founding Time          3-DEC-1992   Largest trans. id        00000466
                         00:01:44   Resource Alloc. retry           0
Index of next CSID           00D2   Figure of Merit          00000000
Quorum Disk Cntrl Block  80334E00   Member State Seq. Num        0190
Timer Entry Address      00000000   Foreign Cluster          00000000
CSP Queue                   empty

         --- Cluster Failover Control Block (CLUFCB) 801C407C ---
Flags: 00000000

Failover Step Index      00000028   CSB of Synchr. System    803686F0
Failover Instance ID     00000466

         --- Cluster Quorum Disk Control Block (CLUDCB) 80334E00 ---
State: 0001 qs_not_ready
Flags: 0000

Iteration Counter               0             UCB address    00000000
Activity Counter                0             TQE address    80419F40
Quorum file LBN          00000000             IRP address    803665A0

         --- SOLLY Cluster System Block (CSB) 803686F0 ---
State: 01 open
Flags: 02020302 member,cluster,qf_active,selected,status_rcvd

Quorum/Votes       2/1   Next seq. number    0247   Send queue      00000000
Quor. Disk Vote      1   Last seq num rcvd   0314   Resend queue    00000000
CSID          000100C8   Last ack. seq num   0247   Block xfer Q.      empty
Eco/Version       0/12   Unacked messages       1   CDT address     801C28F0
Reconn. time  00000059   Ack limit              4   PDT address     801CEA20
Ref. count           2   Incarnation  18-DEC-1993   TQE address     00000000
Ref. time  18-DEC-1993                   08:52:20   SB address      8041B6E0
              08:53:58   Lock mgr dir wgt       1   Current CDRP    00000000
```

This example shows the screen displays for the SHOW CLUSTER command. (Displays for nodes GUS and DORIS, similar to that for node SOLLY, are also included in the SHOW CLUSTER output but have been omitted from this example.)

2. SDA> SHOW CLUSTER /CSID=000100C8

```
VAXcluster data structures
--------------------------
         --- SOLLY Cluster System Block (CSB) 803686F0 ---

State: 01 open
Flags: 02020302 member,cluster,qf_active,selected,status_rcvd

Quorum/Votes       2/1    Next seq. number    0247    Send queue      00000000
Quor. Disk Vote      1    Last seq num rcvd   0314    Resend queue    00000000
CSID          000100C8    Last ack. seq num   0247    Block xfer Q.      empty
Eco/Version       0/12    Unacked messages       1    CDT address     801C28F0
Reconn. time  00000059    Ack limit              4    PDT address     801CEA20
Ref. count           2    Incarnation  18-DEC-1993    TQE address     00000000
Ref. time  18-DEC-1993                     08:52:20    SB address      8041B6E0
              08:53:58    Lock mgr dir wgt       1    Current CDRP    00000000
```

This example shows the use of the /CSID qualifier to obtain information about a specific node (in this instance, node SOLLY).

3. SDA> SHOW CLUSTER /NODE=LEON01

```
VAXcluster data structures
--------------------------

      --- LEON01 Cluster System Block (CSB) 9863BC00 ---

State: 01 open
Status 0206E1A2 member,qf_noaccess,cluster,selected,status_rcvd
                cwps,rangelock,dyn_remaster,dts,vcc
Cpblty 00000001 rm8sec

Quorum/Votes       4/1    Next seq. number    5D8B    Send queue      987C3F80
Quor. Disk Vote     10    Last seq num rcvd   3302    Resend queue    00000000
CSID          00200093    Last ack. seq num   5D8A    Block xfer Q.      empty
Eco/Version       0/24    Unacked messages       0    CDT address     9830C600
Reconn. time  00000000    Ack limit              3    PDT address     98388590
Ref. count           2    Incarnation  26-JAN-1993    TQE address     00000000
Ref. time  26-JAN-1993                     15:14:37    SB address      98638140
              15:28:43    Lock mgr dir wgt       1    Current CDRP    00000000
```

This example shows the use of the /NODE qualifier to obtain information about a specific node (in this instance, node LEON01).

4. SDA> SHOW CLUSTER /SCS

```
VAXcluster data structures
--------------------------

                --- SCS Listening Process Directory ---

Entry Address     Connection ID     Process Name           Information
-------------     -------------     ------------           -----------

  80419D60          08EE0000        SCS$DIRECTORY
  80419E20          08EE0001        VMS$VAXcluster


                  --- SCS Systems Summary ---

  SB Address     Node      Type       System ID         Paths
  ----------     ----      ----       ---------         -----

  8041A120      PINTO      HSC       00000000F10E          1
  8041AA20      DORIS      VMS       0000000008A9          1
  8041AB40      GUS        VMS       0000000008A1          1
  8041B6E0      SOLLY      VMS       0000000008A0          1
  8041D420      DODGER     HSC       00000000F00F          1
```

```
                    --- PINTO System Block (SB) 8041A120 ---

System ID            00000000F10E    Local software type           HSC
Max message size               66    Local software vers.         X301
Max datagram size              62    Local software incarn.   8355FE00
Local hardware type          HS50                             008DA59A
Local hardware vers. 022702220222    SCS poller timeout            000F
                     022202220222    SCS poller enable mask          01

                    --- Path Block (PB) 8041C400 ---

                  Status:  0000

Remote sta. addr.    00000000000E    Remote port type              HSC
Remote state         00000000000E    Number of data paths            2
Remote hardware rev.     00000225    Cables state            A-OK B-OK
Remote func. mask        4F710200    Local state                  OPEN
Resetting port                0E    Port dev. name               PAB0
Handshake retry cnt.           1    SCS MSGBUF address       80390270
Msg. buf. wait queue       empty    PDT address              801CEA20

                    --- DORIS System Block (SB) 8041AA20 ---

System ID            0000000008A9    Local software type           VMS
Max message size              112    Local software vers.         V5.0
Max datagram size             576    Local software incarn.   A9D31760
Local hardware type          V780                             008DA59B
Local hardware vers. 010E0138207A    SCS poller timeout            000C
                     000030030E10    SCS poller enable mask          00

                    --- Path Block (PB) 80437E80 ---

                 Status:  0000

Remote sta. addr.    000000000002    Remote port type             CI780
Remote state                ENAB    Number of data paths            2
Remote hardware rev.     00040003    Cables state            A-OK B-OK
Remote func. mask        FFFFFF00    Local state                  OPEN
Resetting port                02    Port dev. name               PAB0
Handshake retry cnt.           1    SCS MSGBUF address       8036F0B0
Msg. buf. wait queue       empty    PDT address              801CEA20
```

This example shows a subset of a typical output for the SHOW CLUSTER/SCS command. In this system, there are three nodes (DORIS, GUS, and SOLLY), and there are two HSCs (PINTO and DODGER). After the summary information in the first two screen displays, specific information for each system block and its associated path block is shown.

SHOW CONNECTIONS

Displays information about all active connections between systems communications services (SCS) processes or a single connection. This command displays information that is in the connection descriptor table (CDT).

Format

SHOW CONNECTIONS {/ADDR or /ADDRESS=cdt-address | /NODE=name | /SYSAP=name}

Parameters

None.

Qualifiers

/ADDR or /ADDRESS=cdt-address
Displays information contained in the connection descriptor table (CDT) for a specific connection.[9]

/NODE=name
Displays information contained in the connection descriptor table (CDT) for a specific node.

/SYSAP=name
Displays information contained in the connection descriptor table (CDT) for a specific system application (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.

[9] 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 (CDRPs) and unit control blocks (UCBs) for class drivers that use SCS and cluster system blocks (CSBs) for the connection manager.

Examples

1. SDA> SHOW CONNECTIONS

```
VAXcluster data structures
--------------------------

                     --- CDT Summary Page ---

CDT Address   Local Process     Connection ID     State       Remote Node
-----------   -------------     -------------     -----       -----------

 801C2670     SCS$DIRECTORY       08EE0000        listen
 801C2710     VMS$VAXcluster      08EE0001        listen
 801C27B0     VMS$VAXcluster      08FF0002        open         DORIS
 801C2850     VMS$DISK_CL_DRVR    08FD0003        open         PINTO
 801C28F0     VMS$VAXcluster      08EF0004        open         SOLLY
 801C2990     VMS$VAXcluster      08F00005        open         GUS

Number of free CDTs:  32

           --- Connection Descriptor Table (CDT) 801C2670 ---

State:  0001 listen          Local Process:        SCS$DIRECTORY
Blocked State:  0000

Local Con. ID   08EE0000   Datagrams sent    0   Message queue      empty
Remote Con. ID  78A30017   Datagrams rcvd    0   Send Credit Q.     empty
Receive Credit         0   Datagram discard  0   PB address      80438300
Send Credit            1   Messages Sent     0   PDT address     801CEA20
Min. Rec. Credit       0   Messages Rcvd.    0   Error Notify    8022B816
Pend Rec. Credit       0   Send Data Init.   0   Receive Buffer  00000000
Initial Rec. Credit    0   Req Data Init.    0   Connect Data    00000000
Rem. Sta.   00000000000C   Bytes Sent        0   Aux. Structure  00000000
Rej/Disconn Reason     0   Bytes rcvd        0
Queued for BDT         0   Total bytes map   0
Queued Send Credit     0
```

This example shows the CDT summary page and the first page of the detailed displays for each CDT.

2. SDA> SHOW CONNECTIONS /ADDRESS=801C27B0

```
VAXcluster data structures
--------------------------

           --- Connection Descriptor Table (CDT) 801C27B0 ---

State:  0002 open        Local Process:          VMS$VAXcluster
Blocked State:  0000     Remote Node::Process:   DORIS::VMS$VAXcluster

Local Con. ID   08FF0002  Datagrams sent     0   Message queue      empty
Remote Con. ID  33440003  Datagrams rcvd     0   Send Credit Q.     empty
Receive Credit         4  Datagram discard   0   PB address      80437E80
   .
   .
   .
```

This example shows the use of the /ADDRESS qualifier to obtain information about a specific connection.

3. SDA> SHOW CONNECTIONS/NODE=MOON

```
VAXcluster data structures
--------------------------

   --- Connection Descriptor Table (CDT) 98310EE0 ---

State:  0002 open    Local Process:        MSCP$DISK
Blocked State:  0000              Remote Node::Process: MOON::VMS$DISK_CL_DRVR

Local Con. ID   7C79004E    Datagrams sent         0    Message queue      empty
Remote Con. ID  009F0069    Datagrams rcvd         0    Send Credit Q.     empty
Receive Credit        16    Datagram discard       0    PB address      98348200
Send Credit           10    Messages Sent        964    PDT address     98336590
Min. Rec. Credit       1    Messages Rcvd.       808    Error Notify    98B6158D
Pend Rec. Credit       0    Send Data Init.        0    Receive Buffer  986791E8
Initial Rec. Credit   10    Req Data Init.         0    Connect Data    98B60079
Rem. Sta.   000000000009    Bytes Sent             0    Aux. Structure  98679A80
Rej/Disconn Reason     0    Bytes rcvd             0
Queued for BDT         0    Total bytes map        0
Queued Send Credit     0

   --- Connection Descriptor Table (CDT) 98310540 ---

State:  0002 open    Local Process:        SCA$TRANSPORT
Blocked State:  0000              Remote Node::Process: MOON::SCA$TRANSPORT

Local Con. ID   7CCD0047    Datagrams sent         0    Message queue      empty
Remote Con. ID  817F005D    Datagrams rcvd         0    Send Credit Q.     empty
   .
   .
   .

   --- Connection Descriptor Table (CDT) 9830F0A0 ---

State:  0002 open    Local Process:        VMS$DISK_CL_DRVR
Blocked State:  0000              Remote Node::Process: MOON::MSCP$DISK

Local Con. ID   7C790038    Datagrams sent         0    Message queue      empty
Remote Con. ID  4B51005B    Datagrams rcvd         0    Send Credit Q.     empty
   .
   .
   .

   --- Connection Descriptor Table (CDT) 9830EF40 ---

State:  0002 open    Local Process:        VMS$TAPE_CL_DRVR
Blocked State:  0000              Remote Node::Process: MOON::MSCP$TAPE

Local Con. ID   7C790037    Datagrams sent         0    Message queue      empty
Remote Con. ID  23B20068    Datagrams rcvd         0    Send Credit Q.     empty
   .
   .
   .
```

The command in this example displays information in the CDT about the node MOON.

4. SDA> SHOW CONNECTIONS/SYSAP=SCA$TRANSPORT

```
    --- CDT Summary Page ---

CDT Address   Local Process     Connection ID     State       Remote Node
-----------   -------------     -------------     -----       -----------

 9830A7C0     SCA$TRANSPORT       7C790003        listen
 98310540     SCA$TRANSPORT       7CCD0047        open         METEOR
 98310800     SCA$TRANSPORT       7CD50049        open         OCALA

Number of free CDT's:  158

   --- Connection Descriptor Table (CDT) 9830A7C0 ---
```

```
State:  0001 listen    Local Process:        SCA$TRANSPORT
Blocked State:  0000

Local Con. ID   7C790003    Datagrams sent         0    Message queue      empty
Remote Con. ID  00000000    Datagrams rcvd         0    Send Credit Q.     empty
Receive Credit         0    Datagram discard       0    PB address      00000000
Send Credit            0    Messages Sent          0    PDT address     00000000
Min. Rec. Credit       0    Messages Rcvd.         0    Error Notify    968D9E8B
Pend Rec. Credit       0    Send Data Init.        0    Receive Buffer  00000000
Initial Rec. Credit    0    Req Data Init.         0    Connect Data    00000000
Rem. Sta.   000000000000    Bytes Sent             0    Aux. Structure  00000000
Rej/Disconn Reason     0    Bytes rcvd             0
Queued for BDT         0    Total bytes map        0
Queued Send Credit     0

   --- Connection Descriptor Table (CDT) 98310540 ---

State:  0002 open    Local Process:        SCA$TRANSPORT
Blocked State:  0000              Remote Node::Process: METEOR::SCA$TRANSPORT

Local Con. ID   7CCD0047    Datagrams sent         0    Message queue      empty
Remote Con. ID  817F005D    Datagrams rcvd         0    Send Credit Q.     empty
   .
   .
   .

   --- Connection Descriptor Table (CDT) 98310800 ---

State:  0002 open    Local Process:        SCA$TRANSPORT
Blocked State:  0000              Remote Node::Process: OCALA::SCA$TRANSPORT

Local Con. ID   7CD50049    Datagrams sent         0    Message queue      empty
Remote Con. ID  EFB80009    Datagrams rcvd         0    Send Credit Q.     empty
   .
   .
   .
```

This example shows the use of the /SYSAP qualifier to show which nodes in the cluster are connected to SCA$TRANSPORT.

SHOW CPU

Displays information about the state of a processor at the time of the system failure.

Format

SHOW CPU [cpu-id]

Parameter

cpu-id
Numeric value from 00 to $1F_{16}$ indicating the identity of the processor for which context information is to be displayed. 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 CPU command performs an implicit SET CPU command, making the processor indicated by **cpu-id** the current CPU for subsequent SDA commands. (See the description of the SET CPU command and Section 4 for information about how this can affect the CPU context—and process context—in which SDA commands execute.)

Qualifiers

None.

Description

The SHOW CPU command displays crash information about the processor specified by **cpu-id** or, by default, the SDA current CPU, as defined in Section 4. 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 crash 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 processor, 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 processor at the time of the system failure

Next, the **general registers** display shows the contents of the processor's general-purpose registers (R0 through R11) and the AP, FP, SP, PC, and PSL at the time of the crash.

The **processor registers** display consists of the following three parts:

- Common processor registers
- Processor-specific registers

- Stack pointers and memory interconnect silos

The first section includes registers that maintain the virtual address space, system space, or other system functions of the current process. The following registers are among those displayed:

Register	Description
P0BR	Program region (P0 space) base register
P0LR	Program region length register
P1BR	Control region (P1 space) base register
P1LR	Control region length register
SBR	System region (S0 space) base register
SLR	System region length register
PCBB	Process control block base register
SCBB	System control block base register
ASTLVL	Asynchronous system trap level
SISR	Software interrupt summary register
ICCS	Internal clock control and status register
SID	System identification register

The second section of the **processor registers** display shows those registers that are specific to the type of processor being examined. (The SHOW CRASH command displays the processor type.) The contents of the register display vary according to the type of processor involved in the crash and are used primarily in hardware diagnostics.

The final section of the display includes the five stack pointers: the interrupt stack pointer (ISP) and the four pointers of the kernel, executive, supervisor, and user stacks (KSP, ESP, SSP, and USP, respectively). Certain processors, such as the VAX 8800 and VAX 8600 processors, also display the contents of the silos of their memory interconnects in this section.

The SHOW CPU command concludes with a listing of the spin locks, if any, owned by the processor at the time of the crash, reproducing some of the information given by the SHOW SPINLOCKS command. The spin lock display includes the following information:

- Name of the spin lock.
- Address of the spin lock data structure (SPL).
- IPL and rank of the spin lock.
- Number of processors waiting for this processor to release the spin lock.
- Indication of the depth of this processor's ownership of the spin lock. A number greater than 1 indicates that this processor has nested acquisitions of the spin lock.

Example

```
SDA> SHOW CPU

CPU 00 Processor crash information
---------------------------------

CPU 00 reason for Bugcheck: INVEXCEPTN, Exception while above ASTDEL or
on interrupt stack

Process currently executing: NETACP

Current image file: $254$DUA200:[SYS6.SYSCOMMON.]<SYSEXE>NETACP.EXE;3

Current IPL: 8  (decimal)

General registers:

        R0  = 00000008   R1  = 00080000   R2  = 8047FC40   R3  = 000003AC
        R4  = 00000002   R5  = 8047FC40   R6  = 00000036   R7  = 00000000
        R8  = 00000000   R9  = 00000062   R10 = 7FFE7D70   R11 = 0000747C
        AP  = 0000BE34   FP  = 7FFE7DD0   SP  = 7FFE7D30   PC  = 80146682
        PSL = 00080009

Processor registers:

        P0BR   = 816EB600     SBR    = 01A6A800     ASTLVL = 00000004
        P0LR   = 00000C0C     SLR    = 00065600     SISR   = 00000000
        P1BR   = 80FFCE00     PCBB   = 008AF2A0     ICCS   = 00000041
        P1LR   = 001FFC5F     SCBB   = 01A62600     SID    = 067F014F

        ICR    = FFFFEDEA     REVR1  = 11121111     NMIFSR = 000C0000
        TODR   = 2B914C0F     REVR2  = FF00FF12     NMIEAR = 2243F830
        COR    = 00000001     CPUINFO= 000009F7

        ISP    = 8016AC00
        KSP    = 7FFE7D30
        ESP    = 7FFE9E00
        SSP    = 7FFEDE00
        USP    = 7FF8E590

NMI bus silo:

                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
                  00000000
```

```
               Spinlocks currently owned by CPU 00

IOLOCK8                            Address : 80185E50
Owner CPU ID     : 00              IPL     : 08
Ownership Depth  : 0001            Rank    : 14
CPUs Waiting     : 0000            Index   : 34

SDA>  EXAMINE R5
R5:  8047FC40   "@üG."
SDA>  SHOW PROCESS

Process index: 000D   Name: NETACP   Extended PID: 33C0010D
------------------------------------------------------------
Process status:  00148001   RES,NOACNT,PHDRES,LOGIN
   .
   .
   .
SDA>  SHOW CPU 01

CPU 01 Processor crash information
----------------------------------

CPU 01 reason for Bugcheck: CPUEXIT, Shutdown requested by another CPU

Process currently executing: no processes currently scheduled on this CPU

Current IPL: 31  (decimal)
   .
   .
   .
            No spinlocks currently owned by CPU 01

SDA>  EXAMINE R5

R5:  83ED5E00   ".^í."
SDA>  SHOW PROCESS

%SDA-E-BADPROC, no such process
```

This SDA session illustrates the output of the SHOW CPU command in the analysis of a crash dump from a VAX 8800 multiprocessing system with two active processors. The first SHOW CPU command displays the crash information particular to CPU 00, which initially posted an INVEXCEPTN bugcheck from within process NETACP and then requested CPU 01 to take a bugcheck (CPUEXIT) as well. That the crash occurred at IPL 8 signifies, perhaps, that a driver fork process is involved.

The second instance of the SHOW CPU command (SHOW CPU 01) corroborates that CPU 01 was requested to crash by CPU 00.

Significantly, the second SHOW CPU command changes both the SDA current CPU context and current process context. The two EXAMINE R5 commands are executed under different CPU contexts; the values they produce differ. In the CPU context of CPU 00, the current process context is that of process NETACP. There is no current process on CPU 01; thus, SDA process context is initially undefined when its CPU context is changed to that of CPU 01.

SHOW CRASH

In the analysis of a system failure, displays information about the state of the system at the time of the failure. In the analysis of a running system, provides information identifying the system.

Format

SHOW CRASH

Parameters

None.

Qualifiers

None.

Description

The SHOW CRASH command has two different manifestations, depending upon whether you use it while analyzing a running system or a system failure.

In either case, if the SDA current CPU context is not that of the processor that signaled the bugcheck, the SHOW CRASH command performs an implicit SET CPU command to make that processor the SDA current CPU. (See the description of the SET CPU command and Section 4 for a discussion of how this can affect the CPU context—and process context—in which SDA commands execute.)

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 that it is running. The **system crash information** display contains the following information:

- Date and time that the ANALYZE/SYSTEM command was issued (titled "Time of system crash" in the display)
- Name and version number of the operating system
- Major and minor IDs of the operating system
- Identity of the system, including an indication of its VAXcluster membership
- CPU ID of the primary CPU
- Two bit masks indicating which processors in the system are active and which are available for booting, respectively

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.

The **system crash information** display in this context provides the following information:

- Date and time of the system crash.
- Name and version number of the operating system.
- Major and minor IDs of the operating system.

- Identity of the system, including an indication of its VAXcluster membership and the location of the primary CPU in a multiprocessing configuration.
- CPU IDs of both the primary CPU and the CPU that initiated the bugcheck. In a uniprocessor system, these IDs are identical.
- Two bit masks indicating which processors in the system are active and which are available for booting, respectively.
- For each active processor in the system, the name of the bugcheck that caused the 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 spin lock ownership. The first such screen presents information about the processor that caused the crash; others follow according to the numerical order of their CPU IDs.

Examples

1.
```
$ ANALYZE/SYSTEM

OpenVMS VAX System analyzer

SDA> SHOW CRASH

System crash information
------------------------
Time of system crash: 25-FEB-1993 11:18:06.84

Version of system: OpenVMS VAX VERSION 6.0

System Version Major ID/Minor ID: 10/11

VAXcluster node: BIGTOP, a VAX 8800 - primary CPU (left) was booted

Primary CPU ID:  01

Bitmask of CPUs active/available:  00000003/00000003
SDA>  SHOW PROCESS
%SDA-E-BADPROC, no such process
```

   When issued from within the analysis of a running system, the SHOW CRASH command displays the time the ANALYZE/SYSTEM command was issued as the "Time of system crash." The display indicates that the OpenVMS VAX system in use is a VAX 8800 multiprocessing system, the left CPU of which is the primary CPU. The bit mask indicates that there are two processors available and both are running.

   Note that no SDA current process is defined at this time.

2. $ ANALYZE/CRASH SYS$SYSTEM

```
OpenVMS VAX System dump analyzer

Dump taken on 23-FEB-1993 12:44:30.23
INVEXCEPTN, Exception while above ASTDEL or on
          interrupt stack

SDA> SHOW CRASH

System crash information ❶
-----------------------
Time of system crash: 23-FEB-1993 12:44:30.23

Version of system: OpenVMS VAX VERSION 6.0
System Version Major ID/Minor ID: 10/11

VAXcluster node: MOOSE, a VAX 8800 - primary CPU (left) was booted

Crash CPU ID/Primary CPU ID:  00/01

Bitmask of CPUs active/available:  00000003/00000003

CPU bugcheck codes: ❷
   CPU 00 -- INVEXCEPTN, Exception while above ASTDEL or on
             interrupt stack
  1 other -- CPUEXIT, Shutdown requested by another CPU

CPU 00 Processor crash information
----------------------------------

CPU 00 reason for Bugcheck: INVEXCEPTN, Exception while above ASTDEL
or on interrupt stack ❸

Process currently executing on this CPU: NETACP ❸

Current image file: $254$DUA200:[SYS6.SYSCOMMON.][SYSEXE]NETACP.EXE;3

Current IPL: 8  (decimal) ❹

General registers:

    R0  = 00000008   R1  = 00080000   R2  = 8047FC40   R3  = 000003AC
    R4  = 00000002   R5  = 8047FC40   R6  = 00000036   R7  = 00000000
    R8  = 00000000   R9  = 00000062   R10 = 7FFE7D70   R11 = 0000747C
    AP  = 0000BE34   FP  = 7FFE7DD0   SP  = 7FFE7D30   PC  = 80146682
    PSL = 00080009

Processor registers:

    P0BR   = 816EB600     SBR    = 01A6A800     ASTLVL = 00000004
    P0LR   = 00000C0C     SLR    = 00065600     SISR   = 00000000
    P1BR   = 80FFCE00     PCBB   = 008AF2A0     ICCS   = 00000041
    P1LR   = 001FFC5F     SCBB   = 01A62600     SID    = 067F014F

    ICR    = FFFFEDEA     REVR1  = 11121111     NMIFSR = 000C0000
    TODR   = 2B914C0F     REVR2  = FF00FF12     NMIEAR = 2243F830
    COR    = 00000001     CPUINFO= 000009F7     MEMCSR0= 000700F0
         NBIA0 CSR0 = 00203810         NBIA1 CSR0 = 00000000


    ISP    = 8016AC00
    KSP    = 7FFE7D30
    ESP    = 7FFE9E00
    SSP    = 7FFEDE00
    USP    = 7FF8E590

NMI bus silo:
```

```
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000

    Spinlocks currently owned by CPU 00

IOLOCK8                             Address : 80185E50
Owner CPU ID     : 00               IPL     : 08
Ownership Depth  : 0001             Rank    : 14
CPUs Waiting     : 0000             Index   : 34


CPU 01 Processor crash information
----------------------------------

CPU 01 reason for Bugcheck: CPUEXIT, Shutdown requested by another CPU

Process currently executing on this CPU:   None

Current IPL: 31  (decimal)

General registers:

    R0  = 00000020   R1  = 00000000   R2  = 8000CA78   R3  = 80DAF000
    R4  = 80487000   R5  = 83ED5E00   R6  = 7FFA4188   R7  = 7FF28EB8
    R8  = 7FF28E68   R9  = 7FFA2808   R10 = 7FFA4000   R11 = 7FFE0070
    AP  = 7FF28D90   FP  = 7FF28D98   SP  = 80DAFBF8   PC  = 80765465
    PSL = 041F0000

Processor registers:

    P0BR   = 83EE8E00     SBR    = 01A6A800     ASTLVL = 00000004
    P0LR   = 000001C1     SLR    = 00065600     SISR   = 00000000
    P1BR   = 837FA600     PCBB   = 00BB62A0     ICCS   = 00000041
    P1LR   = 001FF935     SCBB   = 01A62600     SID    = 06FF014F

    ICR    = FFFFE7C1     REVR1  = 11121111     NMIFSR = 000C0000
    TODR   = 2B914C0F     REVR2  = FF00FF12     NMIEAR = 24080000
    COR    = 00000001     CPUINFO= 000009F7     MEMCSR0= 000700F0
    NBIA0 CSR0 = 00203810         NBIA1 CSR0 = 00000000

    ISP    = 80DAFBF8
    KSP    = 7FFE7E00
    ESP    = 7FFE9E00
    SSP    = 7FFED04E
    USP    = 7FF28D90

NMI bus silo:
```

```
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000
            00000000

    No spinlocks currently owned by CPU 01
```

This long display reflects the output of the SHOW CRASH command within the analysis of a system failure that occurred on a VAX 8800 multiprocessing system.

The first part of the display includes the following information:

❶ Identification of the system and the version of OpenVMS it was running at the time of the crash.

❷ Indication that the failed processor (CPU 00) was not the primary processor (CPU 01), but requested CPU 01 to take a CPUEXIT bugcheck. (CPU 01 was, in fact, idle at the time of the crash.)

The next part of the display shows information particular to CPU 00:

❸ CPU 00 encountered an INVEXCEPTN bugcheck while executing the NETACP process.

❹ Although the next step in the analysis might be to examine the interrupt stack of CPU 00, the fact that the failure occurred at IPL 8 might indicate that an I/O driver is involved.

At the end of the example, SDA CPU context remains that of CPU 00; its current process context is that of the NETACP process.

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}

Parameter

device-name
Device or devices for which data structures are to be displayed. There are several uses of the **device-name** parameter.

To Display the Structures for . . .	Action
All devices in the system	Do *not* specify a **device-name** (for example, SHOW DEVICE).
A single device	Specify an entire **device-name** (for example, SHOW DEVICE VTA20).
All devices of a certain type on a single controller	Specify only the device type and controller designation (for example, SHOW DEVICE RTA or SHOW DEVICE RTB).
All devices of a certain type on any controller	Specify only the device type (for example, SHOW DEVICE RT).
All devices whose names begin with a certain character or character string	Specify the character or character string (for example, SHOW DEVICE D).
All devices on a single node or HSC	Specify only the node name or HSC name (for example, SHOW DEVICE GREEN$).

In a VAXcluster environment, device information is displayed for each device in the cluster with the specified **device-name**. You can limit the display to those devices that are on a particular node or HSC by specifying the node name or HSC name as part of the **device-name** (for example, GREEN$D or GREEN$DB).

Qualifier

/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 thus 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.

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 list** display to provide a brief summary of the devices for which it renders information in subsequent screens.

Information in the **DDB list** appears in six columns, the contents of which are as follows:

- Address of the device data block (DDB)
- Controller name
- Name of the ancillary control process (ACP) or extended QIO processor (XQP) associated with the device
- Name of the device driver
- Address of the driver prologue table (DPT)
- Size of the 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.

Symbol	Meaning
UCB	Address of unit control block
SB	Address of system block

Symbol	Meaning
ORB	Address of object rights block
DDB	Address of device data block
DDT	Address of driver dispatch table
CRB	Address of channel request block
AMB	Associated mailbox UCB pointer
IRP	Address of I/O request packet
2P_UCB	Address of alternate UCB for dual-pathed device
LNM	Address of logical name block for mailbox
PDT	Address of port descriptor table
CDDB	Address of class driver descriptor block for MSCP-served device
2P_CDDB	Address of alternate CDDB for MSCP-served device
RWAITCNT	Resource wait count for MSCP-served device
VCB	Address of volume control block for mounted device

If you are examining a driver-related crash, you might find it helpful to issue a SHOW STACK command after the appropriate SHOW DEVICE command, examining the stack for any of these symbols. Note, however, that although SHOW DEVICE 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 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 about defining and examining the contents of device data structures.

For a detailed explanation of I/O data structures displayed by SDA, consult the *OpenVMS VAX Device Support Manual*.

Examples

1. SDA>SHOW DEVICE VTA20

```
VTA20 ==> LTA20                          VT200_Series      UCB address:  8042E4C0

Device status:   00010110 online,bsy,deleteucb
Characteristics: 0C040007 rec,ccl,trm,avl,idv,odv
                 00000200 nnm

Owner UIC [000001,000004]   Operation count       5793   ORB address    8042E590
      PID        00010064   Error count             0   DDB address    80CEF2E0
Class/Type          42/6E   Reference count         2   DDT address    807696FB
Def. buf. size         80   BOFF                 0155   CRB address    80BC8B00
DEVDEPEND        180093A0   Byte count           0100   IRP address    80BE2B00
DEVDEPND2        7962100C   SVAPTE           804801C0   I/O wait queue    empty
FLCK/DLCK        00000012   DEVSTS               0000

                                I/O request queue
                                -----------------
```

```
STATE    IRP      PID    MODE CHAN  FUNC    WCB     EFN    AST     IOSB    STATUS
 C   80BE2B00  00010064  E   FFC0  C000  00000000  29  80127458  7FFA800C  0003
     nop bufio,func
```

This example reproduces the SHOW DEVICE display for a single device unit, VTA20. Whereas this display lists information from the UCB for VTA20, 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 sort of information, specify the **device-name** as VTA (for example, SHOW DEVICE VTA).

2. SDA> SHOW DEVICE DU

```
I/O data structures
-------------------
                            DDB list
                            --------

       Address    Controller     ACP       Driver      DPT   DPT size
       -------    ----------     ---       ------      ---   --------

      80D0B3C0    BLUES$DUA     F11XQP    DSDRIVER   807735B0  679D
      8000B2B8    RED$DUA       F11XQP    DSDRIVER   807735B0  679D
      80D0B9C0    RED$DUS       F11XQP    DSDRIVER   807735B0  679D
      80D08BA0    BIGTOP$DUA    F11XQP    DSDRIVER   807735B0  679D
      80D08AE0    TIMEIN$DUA    F11XQP    DSDRIVER   807735B0  679D
   .
   .
   .
Press RETURN for more.
   .
   .
   .
```

This excerpt from the output of the SHOW DEVICE DU command illustrates the format of the **DDB list** display. In this case, the **DDB list** concerns itself with those devices whose device type begins with DU (that is, DUA and DUS). It displays devices of these types attached to various HSCs (RED$ and BLUES$) and systems in a cluster (BIGTOP$ and TIMEIN$).

Following the **DDB list**, SHOW DEVICE DU produces displays for each controller and each unit on each controller, as illustrated in the next example.

3. SDA> SHOW DEVICE DUS

```
I/O data structures
-------------------
                           DDB list
                           --------

     Address    Controller     ACP       Driver      DPT   DPT size
     -------    ----------     ---       ------      ---   --------

     80D0B9C0    RED$DUS       F11XQP    DSDRIVER   807735B0  679D


Controller: RED$DUS
--------------------

                --- LOVE System Block (SB) 80D0C500 ---

System ID            00000000FFF2    Local software type          HSC
Max message size               66    Local software vers.         Y35Q
Max datagram size              62    Local software incarn.   6DF9E6E0
Local hardware type          HS50                             008FCC83
Local hardware vers. 2722722221A3    SCS poller timeout           0002
                     000000272272    SCS poller enable mask         01

                --- Path Block (PB) 80D0BEA0 ---

            Status:  0028

Remote sta. addr.    00000000000B    Remote port type              HSC
Remote state         00000000000B    Number of data paths            2
Remote hardware rev.     00000225    Cables state            A-OK B-OK
Remote func. mask        4F710200    Local state                  OPEN
Resetting port                05    Port dev. name               PAA0
Handshake retry cnt.           1    SCS MSGBUF address       80BCD510
Msg. buf. wait queue       empty    PDT address              803B38D0

            --- Device Data Block (DDB) 80D0B9C0 ---

Driver name         DUDRIVER   Alloc. class        254   DDT address    80773640
ACP ident                F11   SB address     80D0C500
ACP class               PACK   UCB address    803B9C60

        ---  Primary Channel Request Block (CRB) 80BF7000 ---

Reference count       17    Wait queue         empty    Aux. struct.    803B4150
Due time        00012DCC    Timeout rout.   807743D1    Timeout link    8039E03C
IDB address     80D0C440                                Ctrl. init.     80773774
ADP address     80BF7F70

            --- Driver Dispatch Table (DDT) 80773640 ---

Errlog buf sz          0    Diag buf sz          104    FDT size            244
Start I/O       80773B21    Register dump     return    FDT address     80773680
Alt start I/O     return    Unit init       80775970    Mnt verify      80775BC2
Cancel I/O      807763A7    Unsol int       80774602    Cloned UCB        return


RED$DUS3                                RA81             UCB address:  803B9C60

Device status:   00021810 online,valid,unload,lcl_valid
Characteristics: 1C4D4008 dir,fod,shr,avl,mnt,elg,idv,odv,rnd
                 000002A1 clu,mscp,srv,nnm
```

```
Owner UIC [100001,000063]   Operation count      55595   ORB address    803B9D90
      PID        00000000   Error count              0   DDB address    80D0B9C0
Alloc. lock ID   00010161   Reference count          3   DDT address    80773640
Alloc. class          254   Online count             2   VCB address    8044D940
Class/Type          01/15   BOFF                  0000   CRB address    80BF7000
Def. buf. size        512   Byte count            0A00   PDT address    803B38D0
DEVDEPEND        04E00E33   SVAPTE            835C7738   CDDB address   803B4150
DEVDEPND2        00000000   DEVSTS                0004   I/O wait queue    empty
FLCK/DLCK        00000012   RWAITCNT              0000

            --- Primary Class Driver Data Block (CDDB) 803B4150 ---

Status:              1040 alcls_set,bshadow
Controller Flags:    80D6 cf_shadw,cf_mlths,cf_this,cf_misc,cf_attn,cf_replc

Allocation class     254    CDRP Queue      80BD1170    DDB address     8000B2B8
System ID       0000FFF2    Restart Queue      empty    CRB address     80BF7000
                    0000    DAP Count              1    CDDB link       803C01C0
Contrl. ID      0000FFF2    Contr. timeout        75    PDT address     803B38D0
                01010000    Reinit Count           0    Original UCB    00000000
Response ID     00000000    Wait UCB Count         0    UCB chain       803B89A0
MSCP Cmd status FFFFFFFF

    *** I/O request queue is empty ***

            --- Volume Control Block (VCB) 8044D940 ---

Volume: VMSCMSMASTER     Lock name: VMSCMSMASTER
Status:  A0 extfid,system
Status2: 15 writethru,mountver,nohighwater
Shadow status: 21 shadmast,mvbegun

Mount count            1    Rel. volume            0    AQB address     80D0BAE0
Transactions           3    Max. files        111384    RVT address     803B9C60
Free blocks       205989    Rsvd. files            9    FCB queue       80BD87B0
Window size            7    Cluster size           3    Cache blk.      8044DA30
Vol. lock ID    00010167    Def. extend sz.        5    Shadow mem. FL  80CF5C40
Block. lock ID  01A50139    Record size            0    Shadow mem. BL  80CF5BE0
Shadow lock ID  00010168


        --- Shadow set $254$DUS3 member summary ---

Volume: JAZZLORE

Physical unit     Primary path      Secondary path    Member status
-------------     ------------      --------------    -------------
$254$DUA129       RED                 -- none --      Shadow set member
$254$DUA139       RED                 -- none --      Shadow set member
```

```
               --- ACP Queue Block (AQB) 80D0BAE0 ---

ACP requests are serviced by the eXtended Qio Processor (XQP)

Status: 14 defsys,xqioproc

Mount count           56    ACP type           f11v2    Request queue   00000000
                            ACP class              0

    *** ACP request queue is empty ***


RED$DUS5                                RA80              UCB address:  803B9DF0

Device status:   00021810 online,valid,unload,lcl_valid
Characteristics: 1C4D4008 dir,fod,shr,avl,mnt,elg,idv,odv,rnd
                 000002A1 clu,mscp,srv,nnm
   .
   .
   .
```

This example illustrates the output of the command SHOW DEVICE DUS, where two shadow sets (RED$DUS3 and RED$DUS5) are associated with the HSC RED$. There is a controller display for RED$DUS and a unit display for each of the two shadow sets.

SHOW EXECUTIVE

Displays the location and size of each loadable image that makes up the executive.

Format

SHOW EXECUTIVE

Parameters

None.

Qualifiers

None.

Description

The executive consists of a fixed portion and a loadable portion. The fixed portion is known as SYS$SYSTEM:SYS.EXE and consists of three parts:

- System service dispatch vectors
- Universal executive routine vectors
- Globally referenced data cells

The loadable portion consists of a number of independent images that perform the work of the operating system.

The SHOW EXECUTIVE command lists the location and size of each image within the loadable portion of the executive image. It can thus enable you to determine whether a given memory address falls within the range occupied by a particular loadable image. (Table SDA–13 describes the contents of each loadable image.)

By default, SDA displays each location within the loadable portion of the executive as an offset from the beginning of one of the loadable images; for instance, EXCEPTION+00282. Similarly, those symbols that represent system services point to the vector region and not to the system service's loadable code. When tracing the course of a system failure through the listings of modules contained within a given loadable executive image, you might find it useful to load into the SDA symbol table all global symbols and global entry points defined within one or all modules that make up the loadable portion of the executive image. See the description of the READ command for additional information.

The SHOW EXECUTIVE command usually shows all components of the executive image, as illustrated in the following example. In rare circumstances, you might 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. Later, if the system should fail sometime during initialization, it might not have been able to load some of the modules that appear above EXCEPTION in the SHOW EXECUTIVE display (see the example).

Example

```
SDA> SHOW EXECUTIVE

VMS Executive Layout
--------------------
Image                          Base          End           Length

SYSMSG                         8015AA00      80183600      00028C00
RECOVERY_UNIT_SERVICES         80211400      80212000      00000C00
RMS                            80183600      801A7E00      00024800
CPULOA                         801B2800      801B3200      00000A00
LMF$GROUP_TABLE                801B3800      801B3C00      00000400
SYSLICENSE                     801B4000      801B5400      00001400
SYSGETSYI                      801B5A00      801B7000      00001600
SYSDEVICE                      801B7400      801B8A00      00001600
MESSAGE_ROUTINES               801B9000      801BB600      00002600
EXCEPTION                      801CBA00      801D3E00      00008400
LOGICAL_NAMES                  801D4600      801D6000      00001A00
SECURITY                       801D6600      801D7C00      00001600
LOCKING                        801D8200      801DA800      00002600
PAGE_MANAGEMENT                801DAE00      801E2600      00007800
WORKING_SET_MANAGEMENT         801E2E00      801E7200      00004400
IMAGE_MANAGEMENT               801E7C00      801EA400      00002800
EVENT_FLAGS_AND_ASTS           801EAA00      801EBE00      00001400
IO_ROUTINES                    801EC400      801F2C00      00006800
PROCESS_MANAGEMENT             801F3200      801F9400      00006200
ERRORLOG                       80204C00      80205600      00000A00
PRIMITIVE_IO                   80205C00      80206C00      00001000
SYSTEM_SYNCHRONIZATION         80207000      80208C00      00001C00
SYSTEM_PRIMITIVES              80209200      8020C400      00003200
```

The SHOW EXECUTIVE command displays the location and length of the loadable images included in the executive.

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; similarly, the ninth column, when read left to right, records the ASCII equivalent of the contents. (Note that the period character [.] in this column indicates an ASCII character that cannot be displayed.)

After it displays the contents of the first header block, the SHOW HEADER command displays the hexadecimal contents of the saved error log buffers.

See the *VAX/VMS Internals and Data Structures* manual for a discussion of the information contained in the dump file header.

```
SDA>  SHOW HEADER

Dump file header
----------------
7FF03944 7FFED04E . . . 000000C1 00000000  ........................N...D9.. 00000000
00000000 00000000 . . . 00040000 80185200  .R.............................. 00000020
00000000 00000000 . . . 00000000 00000000  ................................ 00000040
00020000 00000000 . . . 15000011 00000000  ................................ 00000060
414E454C 45480800 . . . 0000012C 00000000  ....,.....................GARNER 00000080
FE9E007F F74D7C0A . . . 00000000 00002020  ...........%.@.041......M....... 000000A0
   .
   .
   .
Saved error log messages
------------------------
00000000 00000009 . . . 801D8739 00000300  ....9.......5................... 801D8600
7B0090AC 2FCBCEC2 . . . 414E454C 45480800  ..GARNER  ...........&.zxcv.O... 801D8620
00202041 4E454C45 . . . 01080100 0000C30A  .A........d............GARNER  . 801D8640
   .
   .
   .
```

The SHOW HEADER command displays the contents of the dump file's header from address $6B0_{16}$ to address $C90_{16}$. Ellipses indicate hexadecimal information omitted from the display.

SHOW LAN

Displays information contained in various local area network (LAN) data structures. The default qualifiers are /CSMACD/FDDI.

Format

SHOW LAN [/qualifier[,...]]

Parameters

None.

Qualifiers

/CLIENT=xx
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. /CLIENT, /DEVICE, and /UNIT are synonymous and mutually exclusive; each must be the last qualifier stated on an SDA command line.

/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 (CSMACD) information for the LAN be displayed.

/CSMACD/FDDI (default)
Displays both Ethernet and FDDI information.

/DEVICE=xx[dn]
Specifies that information be displayed for the specified device. 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. /CLIENT, /DEVICE, and /UNIT are synonymous and mutually exclusive; each must be the last qualifier stated on an SDA command line.

/ERRORS
Specifies that the LSB and UCB error counters be displayed.

/FDDI
Specifies that Fiber Distributed Data Interface (FDDI) controller information for the LAN be displayed.

/FULL
Specifies that all information from the LAN, LSB, and UCB data structures be displayed.

/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 to print time information (start and stop times and error times) from the device and unit data structures. SDA displays the data in chronological order.

/UNIT=xx/[dn]
Specifies that information be displayed for the specified unit. Unit designators are specified in the format **xx/[dn]**, where **xx** is the type of unit, **d** is the device letter, and **n** is the unit number. The device letter and unit number are optional. /CLIENT, /DEVICE, and /UNIT are synonymous and mutually exclusive; each must be the last qualifier stated on an SDA command line.

Description

The SHOW LAN command displays information contained in various local area network (LAN) data structures. By default, or when you specify the /SUMMARY qualifier, SHOW LAN displays a list of flags, LSBs, UCBs, and base addresses. When you specify the /FULL qualifier, SHOW LAN displays all information found in the LAN, LSB, and UCB data structures.

Examples

1.
```
SDA> SHOW LAN

                -- LAN Device Summary 26-JAN-1993 20:57:41 --

LAN block address  = 9834C680  (6 stations)
LAN flags:  0002 LAN_init

LSB address  = 98358B40
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                 -- EXA Unit Summary 26-JAN-1993 20:57:41 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
EXA0    98358540
EXA1    98376340  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
EXA3    98ACD240  Eth   60-03           DECNET  0004 Uniq
EXA5    983A9580  Eth   80-41           LAST    0015 Strtn,Uniq,Strtd

LSB address  = 98369B40
Device state = 4013 Inited,Run,Timer

                 -- FXA Unit Summary 26-JAN-1993 20:57:41 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
FXA0    98369840
FXA1    98391980  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
FXA2    98AC9680  Eth   60-03           DECNET  0017 Strtn,Len,Uniq,Strtd
FXA3    98AC7100  Eth   60-01           MOPDL   001F Strtn,Uniq,Share,Strtd
FXA4    98AC9B80  Eth   90-00           LOOP    001D Strtn,Uniq,Share,Strtd
FXA5    98395380  Eth   60-04           LAT     0015 Strtn,Uniq,Strtd

LSB address  = 9836CE00
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                 -- EXB Unit Summary 26-JAN-1993 20:57:41 --
```

```
UCB     UCB Addr  Fmt   Value          Client     State
---     --------  ---   -----          ------  -----------
EXB0    98358880
EXB1    983B8B00  Eth   60-07          SCA     0017 Strtn,Len,Uniq,Strtd
EXB2    98ACD500  Eth   60-03          DECNET  0004 Uniq

LSB address  = 9836FE00
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                -- EXC Unit Summary 26-JAN-1993 20:57:41 --

UCB     UCB Addr  Fmt   Value          Client     State
---     --------  ---   -----          ------  -----------
EXC0    9836CA80
EXC1    983C08C0  Eth   60-07          SCA     0017 Strtn,Len,Uniq,Strtd
EXC2    98ACD7C0  Eth   60-03          DECNET  0004 Uniq

LSB address  = 98376600
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                -- EXD Unit Summary 26-JAN-1993 20:57:41 --

UCB     UCB Addr  Fmt   Value          Client     State
---     --------  ---   -----          ------  -----------
EXD0    9836FA80
EXD1    983C8680  Eth   60-07          SCA     0017 Strtn,Len,Uniq,Strtd
EXD2    98ACDA80  Eth   60-03          DECNET  0004 Uniq

LSB address  = 98378340
Device state = 4013 Inited,Run,Timer

                -- FXB Unit Summary 26-JAN-1993 20:57:41 --

UCB     UCB Addr  Fmt   Value          Client     State
---     --------  ---   -----          ------  -----------
FXB0    98377F80
FXB1    983D0440  Eth   60-07          SCA     0017 Strtn,Len,Uniq,Strtd
FXB2    98AC9900  Eth   60-03          DECNET  0004 Uniq
```

The SHOW LAN command in this example displays information about LAN data structures, including CSMACD and FDDI information.

2.
```
SDA>  SHOW LAN/COUNTERS/DEV=DECNET

     -- EZA1 60-03 (DECNET) Counters Information 19-JUL-1993 14:27:02 --

Last receive                    None    Last transmit        19-JUL 14:26:51
Octets received               580539    Octets sent               2399353240
PDUs received                   8194    PDUs sent                       5618
Mcast octets received              0    Mcast octets sent                  0
Mcast PDUs received                0    Mcast PDUs sent                    0
Unavail user buffer                0    Last start attempt              None
Last start done      19-JUL 06:40:22    Last start failed               None
```

The SHOW LAN command in this example displays the counters for device DECNET.

3.
```
SDA> SHOW LAN/CSMACD

                -- LAN Device Summary 26-JAN-1993 20:57:22 --

LAN block address  = 9834C680  (6 stations)
LAN flags:  0002 LAN_init

LSB address  = 98358B40
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                -- EXA Unit Summary 26-JAN-1993 20:57:22 --
```

```
UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
EXA0    98358540
EXA1    98376340  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
EXA3    98ACD240  Eth   60-03           DECNET  0004 Uniq
EXA5    983A9580  Eth   80-41           LAST    0015 Strtn,Uniq,Strtd

LSB address  = 9836CE00
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                -- EXB Unit Summary 26-JAN-1993 20:57:22 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
EXB0    98358880
EXB1    983B8B00  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
EXB2    98ACD500  Eth   60-03           DECNET  0004 Uniq

LSB address  = 9836FE00
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                -- EXC Unit Summary 26-JAN-1993 20:57:22 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
EXC0    9836CA80
EXC1    983C08C0  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
EXC2    98ACD7C0  Eth   60-03           DECNET  0004 Uniq

LSB address  = 98376600
Device state = 001B Inited,Run,Ctl_Rdy,Timer

                -- EXD Unit Summary 26-JAN-1993 20:57:22 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
EXD0    9836FA80
EXD1    983C8680  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
EXD2    98ACDA80  Eth   60-03           DECNET  0004 Uniq
```

The SHOW LAN command in this example displays CSMACD information for the LAN.

4. SDA SHOW LAN/FDDI

```
                -- LAN Device Summary 26-JAN-1993 20:57:07 --

LAN block address  = 9834C680  (6 stations)
LAN flags:  0002 LAN_init

LSB address  = 98369B40
Device state = 4013 Inited,Run,Timer

                -- FXA Unit Summary 26-JAN-1993 20:57:07 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
FXA0    98369840
FXA1    98391980  Eth   60-07           SCA     0017 Strtn,Len,Uniq,Strtd
FXA2    98AC9680  Eth   60-03           DECNET  0017 Strtn,Len,Uniq,Strtd
FXA3    98AC7100  Eth   60-01           MOPDL   001F Strtn,Uniq,Share,Strtd
FXA4    98AC9B80  Eth   90-00           LOOP    001D Strtn,Uniq,Share,Strtd
FXA5    98395380  Eth   60-04           LAT     0015 Strtn,Uniq,Strtd

LSB address  = 98378340
Device state = 4013 Inited,Run,Timer

                -- FXB Unit Summary 26-JAN-1993 20:57:07 --
```

```
UCB     UCB Addr  Fmt   Value          Client     State
---     --------  ---   -----          ------  -----------
FXB0    98377F80
FXB1    983D0440  Eth   60-07          SCA     0017 Strtn,Len,Uniq,Strtd
FXB2    98AC9900  Eth   60-03          DECNET  0004 Uniq
```

The SHOW LAN command in this example displays FDDI information.

5. SDA> SHOW LAN/FULL

```
LAN Data Structures

            -- LAN Information Summary 27-JAN-1993 09:54:50 --

LAN flags:  0002 LAN_init

LAN module version                1    First SVAPTE                81FAFC14
LAN address                80EA8C00    Number of PTEs                     4
Number of stations                1    SVA of pages                80A00A00
First LSB address          80ECE700

          -- LAN CSMACD Network Management 27-JAN-1993 09:54:50 --

Creation time                   None    Times created                     0
Deletion time                   None    Times deleted                     0
Module EAB                 00000000    Latest EIB                 00000000
Port EAB                   00000000
Station EAB                00000000

           -- LAN FDDI Network Management 27-JAN-1993 09:54:50 --

Creation time                   None    Times created                     0
Deletion time                   None    Times deleted                     0
Module EAB                 00000000    Latest EIB                 00000000
Port EAB                   00000000
Station EAB                00000000
Link EAB                   00000000
PHY port EAB               00000000

              -- ESA Device Information 27-JAN-1993 09:54:50 --

LSB address                80ECE700    Active unit count                 2
LAN version       00000001 06000036    Driver version     00000001 06000009
LAN code address           80EC8BF9    Driver code address         80EC68B0
Device name                ES_LANCE    Device type                      24
Device version    00000000 00000000    DLL type                      CSMACD

Data chaining                   ON    All multicast state              OFF
Controller mode             NORMAL    Promiscuous mode                 OFF
CRC generation mode             ON    Hardware mode                   0000
Physical address   AA-00-04-00-50-FD    Hardware address   08-00-2B-2A-D7-F7

Flags:  0000                         Characteristics: 0000
Status: 0013 Inited,Run,Timer

DAT stage                  00000000    DAT xmt status     0000001A 001A0001
DAT number started                1    DAT xmt complete     26-JAN 13:20:31
DAT number failed                 0    DAT rcv found                  None
Creation time                  None    Create count                      0
Deletion time                  None    Enable count                      0
Enabled time                   None    Fatal error count                 0
Disabled time                  None    Excessive collisons               0

Last receive         27-JAN 09:54:50    Last fatal error               None
Last transmit        27-JAN 09:54:47    Prev fatal error               None
Last fork sched      27-JAN 09:54:50    Last exc collision   26-JAN 16:36:26
Last fork time       27-JAN 09:54:50
```

```
Rcv buffers owned by device        9    System buffer quota               0
Xmt entries owned by device        0    Device dependent longword   00000000
Xmt entries owned by host          0    # restarts pending                0

NMgmt advised buffer count         0    Events logged                     0
EIB address                 00000000    NMgmt assigned adr 00-00-00-00-00-00
LPB address                 00000000

             -- ESA Queue Information 27-JAN-1993 09:54:50 --

Control hold queue      80ECE820  Status:  Valid, empty
Control request queue   80ECE828  Status:  Valid, empty
Control pending queue   80ECE830  Status:  Valid, empty
Transmit request queue  80ECE818  Status:  Valid, empty
Transmit pending queue  80ECE838  Status:  Valid, empty
Receive buffer queue    80ECE840  Status:  Valid, empty
Receive pending queue   80ECE848  Status:  Valid, 9 elements
Post process queue      80ECE850  Status:  Valid, empty
Delay queue             80ECE858  Status:  Valid, empty
Auto restart queue      80ECE860  Status:  Valid, empty
Netwrk mgmt hold queue  80ECE868  Status:  Valid, empty

        -- ESA Multicast Address Information 27-JAN-1993 09:54:50 --

AB-00-00-04-00-00
09-00-2B-04-00-00

                -- ESA Unit Summary 27-JAN-1993 09:54:50 --

UCB     UCB Addr  Fmt   Value           Client     State
---     --------  ---   -----           ------  -----------
ESA0    80EC61C0
ESA2    80EFD600  Eth   60-03           DECNET  0017 Strtn,Len,Uniq,Strtd
ESA4    80F505C0  Eth   80-41           LAST    0015 Strtn,Uniq,Strtd

  .
  .
  .

        -- ESA Internal Counters Information 27-JAN-1993 09:54:50 --

Internal counters address   80ECF6E8    Internal counters size            30
Number of ports                    0    Global page transmits             0
No work transmits                  0    SVAPTE/BOFF transmits             0
Bad PTE transmits                  0    Buffer_Adr transmits              0

Fatal error count                  0    RDL errors                        0
Transmit timeouts                  0    Last fatal error               None
Restart failures                   0    Prev fatal error               None
Power failures                     0    Last error CSR             00000000
Hardware errors                    0    Fatal error code               None
Control timeouts                   0    Prev fatal error               None

Loopback sent                      0    Loopback failures                 0
System ID sent                   121    System ID failures                0
ReqCounters sent                   0    ReqCounters failures              0

          -- ESA0 Template Unit Information 27-JAN-1993 09:54:50 --
```

```
LSB address                80ECE700    VCIB address                00000000
Packet format              Ethernet    Error count                        0
Device buffer size             1500    LAN medium                     CSMACD
Maximum buffer size            1500    Eth protocol type               00-00
Hardware buffer quota             9    802E protocol ID      00-00-00-00-00
Receive buffer quota              0    802.2 SAP                         00
Allow prom client                ON    802.2 Group SAPs         00,00,00,00
Promiscuous mode                OFF    Maximum header size                0
802.2 service                   OFF    Hardware address   08-00-2B-2A-D7-F7
Data chaining                   OFF    Physical address   FF-FF-FF-FF-FF-FF
Padding mode                     ON    Can change address               OFF
Automatic restart               OFF    Access mode                EXCLUSIVE
CRC generation mode              ON    Controller mode               NORMAL
Maintenance state                ON    Rcv buffs to queue                 1
P2 parameters              00000000    Starter's PID               00000000
All multicast mode              OFF    Creator's PID               00000000
Rcv buffer quota                  0    LSB size                        5986

       -- ESA2 60-03 (DECNET) Unit Information 27-JAN-1993 09:54:50 --

LSB address                80ECE700    VCIB address                00000000
Packet format              Ethernet    Error count                        0
Device buffer size             1500    LAN medium                     CSMACD
Maximum buffer size            1498    Eth protocol type               60-03
Hardware buffer quota             9    802E protocol ID      00-00-00-00-00
Receive buffer quota          15040    802.2 SAP                         00
Allow prom client                ON    802.2 Group SAPs         00,00,00,00
Promiscuous mode                OFF    Maximum header size               16
802.2 service                   OFF    Hardware address   08-00-2B-2A-D7-F7
Data chaining                   OFF    Physical address   AA-00-04-00-50-FD
Padding mode                     ON    Can change address               OFF
Automatic restart               OFF    Access mode                EXCLUSIVE
CRC generation mode              ON    Controller mode               NORMAL
Maintenance state                ON    Rcv buffs to queue                10
P2 parameters              00374395    Starter's PID               0001000C
All multicast mode              OFF    Creator's PID               0001000C
Rcv buffer quota              15040    LSB size                        5986

     -- ESA2 60-03 (DECNET) Counters & Misc Info 27-JAN-1993 09:54:50 --

Last receive         27-JAN 09:54:50    Last transmit        27-JAN 09:54:47
Octets received             5087025    Octets sent                 2310540
PDUs received                 34018    PDUs sent                     29121
Mcast octets received        2189558    Mcast octets sent            246850
Mcast PDUs received             9877    Mcast PDUs sent                4937
Unavail user buffer               11    Last start attempt             None
Last start done      26-JAN 13:20:32    Last start failed              None
                                        Share UCB total quota             0

Receive IRP queue       80EFD7C4  Status:  Valid, 1 element
Shared users queue      80EFD7B4  Status:  Valid, empty
Receive pending queue   80EFD7BC  Status:  Valid, empty

    -- ESA2 60-03 (DECNET) Multicast Address Info 27-JAN-1993 09:54:50 --

Multicast address table, embedded:
  AB-00-00-04-00-00


       -- ESA4 80-41 (LAST) Unit Information 27-JAN-1993 09:54:50 --

LSB address                80ECE700    VCIB address                80F504F3
Packet format              Ethernet    Error count                        0
   .
   .
   .
```

```
      -- ESA4 80-41 (LAST) Counters & Misc Info 27-JAN-1993 09:54:50 --

Last receive         27-JAN 09:54:39    Last transmit        27-JAN 09:54:38
Octets received              1941967    Octets sent                  371740
   .
   .
   .
     -- ESA4 80-41 (LAST) Multicast Address Info 27-JAN-1993 09:54:50 --

Multicast address table, embedded:
  09-00-2B-04-00-00
```

The SHOW LAN/FULL command in this example displays information for all LAN, LSB, and UCB data structures.

6.
```
SDA> SHOW LAN/TIMESTAMPS

LAN Data Structures
-------------------
             -- LAN History Information 19-JUL-1993 14:27:38 --

19-JUL 14:27:38.93  EZA               Last receive
19-JUL 14:27:38.93  EZA               Last fork scheduled
19-JUL 14:27:38.93  EZA               Last fork time
19-JUL 14:27:36.05  EZA               Last transmit
19-JUL 14:27:36.05  EZA1     DECNET   Last transmit
19-JUL 14:23:54.41  EZA164   DIAG     Last start completed
19-JUL 08:05:16.09  EZA               Last excessive collision
19-JUL 06:40:22.94  EZA1     DECNET   Last start completed
19-JUL 06:40:21.94  EZA               Last DAT transmit
```

The SHOW LAN command displays LAN timestamp information.

SHOW LOCK

Displays information about all lock management locks in the system, cached locks, or a specified lock.

Format

SHOW LOCK {lock-id | /ALL | /CACHED | /NAME=resource-name}

Parameters

lock-id
Name of a specific lock. You cannot specify both a **lock-id** and a **resource-name** in the same command line.

Qualifiers

/ALL
Lists all locks that exist in the system. This is the default behavior of the SHOW LOCK command.

/CACHED
Shows only cached lock blocks (LKBs).

/NAME=resource-name
Displays information about the resource associated with the lock whose resource name begins with the specified **resource-name**. For case-sensitive names, enclose the **resource-name** in quotation marks. You cannot specify both a **lock-id** and **resource-name** in the same command line.

Description

The SHOW LOCK command displays the information described in Table SDA–15 for each lock management lock in the system or for the lock indicated by **lock-id**. (Use the SHOW SPINLOCK command to display information about spin locks.) 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 System Services Reference Manual* for additional discussion of the significance of this information.

You can display information about the resource to which a lock is queued by issuing the SHOW RESOURCE command and specifying the **lock-id** of the resource.

Table SDA–15 Contents of the SHOW LOCK and SHOW PROCESS/LOCKS Displays

Display Element	Description
Process Index[1]	Index into the PCB array to a pointer to the process control block (PCB) of the process that owns the lock.
Name[1]	Name of the process that owns the lock.
Extended PID[1]	Clusterwide identification of the process that owns the lock.
Lock ID	Identification of the lock.
PID	Systemwide identification of the lock.
Flags	Information specified in the request for the lock.
Par. ID	Identification of the lock's parent lock.
Granted at	Lock mode at which the lock was granted.
Sublocks	Identification numbers of the locks that the lock owns.
LKB	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.
Resource	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.
Status	Status of the lock, information used internally by the lock manager.
Length	Length of the resource name.
—	Processor access mode of the name space in which the resource block (RSB) associated with the lock resides.
—	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.
—	Indication of whether the lock is mastered on the local system or is a process copy.

[1]You produce this display element only by using the SHOW PROCESS/LOCKS command.

Examples

1.
```
SDA> SHOW LOCK

Lock database
-------------

Lock id:  00010001   PID:    00000000   Flags:   NOQUEUE SYNCSTS SYSTEM
Par. id:  00000000   Granted at    EX            CVTSYS
Sublocks:        1
LKB:      80D0B8A0
Resource:      5F535953 24535953    SYS$SYS_  Status:  NOQUOTA
 Length   16   00000000 4C774449    IDwL....
 Exec. mode    00000000 00000000    ........
 System        00000000 00000000    ........
Local copy


Lock id:  00010004   PID:    00000000   Flags:   CONVERT SYNCSTS CVTSYS
Par. id:  00000000   Granted at    CR
Sublocks:       16
LKB:      80D091A0   BLKAST
Resource:      4D567624 42313146    F11B$vVM  Status:  NOQUOTA
 Length   18   20204E41 4A353153    S15JAN
 Kernel mode   00000000 00002020      ......
 System        00000000 00000000    ........
Local copy


Lock id:  00280009   PID:    00000000   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  00000000   Granted at    CR            NOQUOTA CVTSYS
Sublocks:        0
LKB:      80CDA880
Resource:      52414B5F 24535953    SYS$_KAR  Status:  MSTCPY
 Length   17   30415544 24455441    ATE$DUA0
 Kernel mode   00000000 0000003A    :.......
 System        00000000 00000000    ........
Master copy of lock 001C00F5 on system 000100A1
   .
   .
   .
SDA>  SHOW RESOURCE/LOCK=280009

Resource database
-----------------
Address of RSB:  80BD2150  Group grant mode:       CR
Parent RSB:      00000000  Conversion grant mode:  CR
Sub-RSB count:          0  BLKAST count:            0
Value block:   00000000 00000000 00000000 00000019   Seq. #:  0000002D
Resource:      52414B5F 24535953    SYS$_KAR
 Length   17   30415544 24455441    ATE$DUA0         CSID:  00000000
 Kernel mode   00000000 0000003A    :.......
 System        00000000 00000000    ........

Granted queue (Lock ID / Gr mode):
     00DA1269  CR       00280009  CR       0094054D  CR
     00270B9F  CR       00D70BFE  CR       000D0F4F  CR
     000D1017  CR       00601418  CR       01131450  CR
     000F1964  CR       000200DF  CR

Conversion queue (Lock ID / Gr/Rq mode):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode):
     *** EMPTY QUEUE ***
```

This SDA session shows the output of the SHOW LOCK command for several locks. The SHOW RESOURCE command, executed for the last displayed lock, verifies that the lock is in the resource's granted queue, among many other locks given concurrent read (CR) access to the resource. (See Table SDA–21 for a full explanation of the contents of the display of the SHOW RESOURCE command.)

2. SDA SHOW LOCK/CACHE

```
Lock database
-------------

Lock id:  6D000032   PID:     00010028   Flags:   VALBLK  SYNCSTS SYSTEM
Par. id:  01000002   SUBLCKs:        0            NOQUOTA
LKB:   80F67C00   BLKAST:  00000000
PRIORTY:      0000

Granted at      PW   00000000-FFFFFFFF

Resource:      00257324 42313146    F11B$s%.  Status:  NOQUOTA CACHED
 Length   10   00000000 00000000    ........
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........

Local copy

Lock id:  7B00003B   PID:     0001000B   Flags:   VALBLK  SYNCSTS SYSTEM
Par. id:  01000002   SUBLCKs:        0            NOQUOTA
LKB:   80F51F80   BLKAST:  00000000
PRIORTY:      0000

Granted at      PW   00000000-FFFFFFFF

Resource:      08E97324 42313146    F11B$sé.  Status:  NOQUOTA CACHED
 Length   10   00000000 00000000    ........
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........

   .
   .
   .
Local copy
```

This example of the SHOW LOCK/CACHE command displays the contents of cached lock blocks (LKBs).

SHOW LOGS

Displays information about transaction logs currently open for the node.

Format

SHOW LOGS [/qualifier[,...]]

Qualifier

/DISPLAY=(item [,...])
Specifies the type of information to be displayed. The argument to /DISPLAY can be either a single item or a list. The following items can be specified.

Item	Description
ALL	All transaction log control structure information. This is the default behavior.
OPENS	Transaction log open requests.
READS	Transaction log read requests.
WRITES	Transaction log write requests.

Example

```
SDA> SHOW LOGS/DISPLAY=(OPENS, WRITES)
```

The SHOW LOGS command displays the log open request and log write request information for all open transaction logs for the node.

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 [/qualifier[,...]] [range]

Parameter

range
Range of virtual addresses for which SDA is to display page table entries. You can express a range using the following format:

m:n Range of virtual addresses from *m* to *n*

m;n Range of virtual addresses starting at *m* and continuing for *n* bytes

Qualifiers

/GLOBAL
Lists the global page table.

/SYSTEM
Lists the system page table.

/ALL
Lists both the global and system page tables. This is the default behavior of SHOW PAGE_TABLE.

Description

For each virtual address displayed by the SHOW PAGE_TABLE command, the first six columns of the listing provide the associated page table entry and describe its location, characteristics, and contents (see Table SDA–16). SDA obtains this information from the system page table.

If the virtual page has been mapped to a physical page, the last nine columns of the listing include information from the page frame number (PFN) database (see Table SDA–17). Otherwise, the section is left blank.

SDA indicates pages are inaccessible by displaying the following message:

```
-------- n NULL PAGES
```

Here, *n* indicates the number of inaccessible pages.

Table SDA–16 Virtual Page Information in the SHOW PAGE_TABLE Display

Value	Meaning
ADDRESS	System virtual address that marks the base of the virtual page.
SVAPTE	System virtual address of the page table entry that maps the virtual page.

(continued on next page)

Table SDA–16 (Cont.) Virtual Page Information in the SHOW PAGE_TABLE Display

Value	Meaning
PTE	Contents of the page table entry, a longword that describes a system virtual page.
Type	Type of virtual page. There are the following eight types: • VALID Valid page (in main memory). • TRANS Transitional page (between main memory and page lists). • DZERO Demand-allocated, zero-filled page. • PGFIL Page within a paging file. • STX Section table's index page. • GPTX Index page for a global page table. • IOPAG Page in I/O address space. • NXMEM 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.
PROT	Protection code, derived from bits in the PTE, that designates the type of access (read or write, or both) granted to processor access modes (kernel, executive, supervisor, or user).

(continued on next page)

Table SDA–16 (Cont.) Virtual Page Information in the SHOW PAGE_TABLE Display

Value	Meaning
Bits	Letters that represent the setting of a bit or a combination of bits in the PTE. These bits indicate attributes of a page. The following codes are listed: • M Page has been modified. • L Page is locked into a working set. • K Owner can access the page in kernel mode. • E Owner can access the page in executive mode. • S Owner can access the page in supervisor mode. • U Owner can access the page in user mode.

Table SDA–17 Physical Page Information in the SHOW PAGE_TABLE Display

Category	Meaning
PAGTYP	Type of physical page. One of the following six types: • PROCESS Page is part of process space. • SYSTEM Page is part of system space. • GLOBAL Page is part of a global section. • PPGTBL Page is part of a process's page table. • GPGTBL Page is part of a global page table. • GBLWRT Page is part of a global, writable section.

(continued on next page)

Table SDA–17 (Cont.) Physical Page Information in the SHOW PAGE_TABLE Display

Category	Meaning
LOC	Location of the page within the system. One of the following eight locations: • ACTIVE Page is in a working set. • MDFYLST Page is in the modified page list. • FREELST Page is in the free page list. • BADLST Page is in the bad page list. • RELPEND Release of the page is pending. • RDERROR Page has had an error during an attempted read operation. • PAGEOUT Page is being written into a paging file. • PAGEIN Page is being brought into memory from a paging file.
STATE	Byte that describes the state of the physical page.
TYPE	Byte that describes the type of virtual page. The types in this column are the hexadecimal codes that stand for the page types that appear in column PAGTYP of this display, described previously.
REFCOUNT	Count of the processes that are referencing this PFN. If the value of REFCOUNT is nonzero, the page is used in at least one working set. If the value is zero, the page is not used in any working set.
BAK	Address of the backing store; location on a disk device to which pages can be written.
SVAPTE	Virtual address associated with this page frame. The two SVAPTEs indicate a valid link between physical and virtual address space.
FLINK	Forward link within PFN database that points to the next virtual page. This longword also acts as the count of the number of processes that are sharing this global section.
BLINK	Backward link within PFN database. Also acts as an index into the working set list.

Example

```
SDA>SHOW PAGE_TABLE
System page table
-----------------
ADDRESS  SVAPTE   PTE      TYPE  PROT  BITS PAGTYP LOC  STATE TYPE REFCNT BAK     SVAPTE    FLINK      BLINK
   .
   .
   .
8014B000 8AD22E00 F8020725 VALID UR       K
8014B200 8AD22E04 F8020726 VALID UR       K
8014B400 8AD22E08 F8020727 VALID UR       K
8014B600 8AD22E0C F8020728 VALID UR       K
8014B800 8AD22E10 F8020729 VALID UR       K
8014BA00 8AD22E14 EC02072A VALID UREW  M  K
8014BC00 8AD22E18 F402072B VALID URKW  M  K
   .
   .
   .
8014BE00 8AD22FEC F801F10E VALID UR       K SYSTEM ACTIVE  07  01   1 0040FFF8   8AD22FEC  00000000  00000258
8014C000 8AD22FF0 F801F10F VALID UR       K SYSTEM ACTIVE  07  01   1 0040FFF8   8AD22FF0  00000000  00000257
8014C200 8AD22FF4 F801F173 VALID UR       K SYSTEM ACTIVE  07  01   1 0040FFF8   8AD22FF4  00000000  000004B1
8014C400 8AD22FF8 F801F172 VALID UR       K SYSTEM ACTIVE  07  01   1 0040FFF8   8AD22FF8  00000000  00000301
8014C600 8AD22FFC F801F17F VALID UR       K SYSTEM ACTIVE  07  01   1 0040FFF8   8AD22FFC  00000000  000000F5
8014C800 8AD23000 F801F17E VALID UR       K SYSTEM ACTIVE  07  01   1 0040FFF8   8AD23000  00000000  00000174
8014CA00 8AD23004 7801EBC6 TRANS UR       K SYSTEM FREELST 00  01   0 0040FFF8   8AD23004  0000D38B  0001EBC7
   .
   .
   .
```

SHOW PFN_DATA

Displays information that is contained in the page lists and PFN database.

Format

SHOW PFN_DATA [pfn] [/qualifier]

Parameter

pfn
Page frame number (PFN) of the physical page for which information is to be displayed.

Qualifiers

/ALL
Displays the free page list, modified page list, and bad page list. 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.

/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.

/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.

/SYSTEM
Displays the entire PFN database in order by page frame number, starting at PFN 0000.

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. Table SDA–18 lists the contents of the display.

Table SDA–18 Page Frame Number Information in the SHOW PFN_DATA Display

Item	Contents
PFN	Page frame number
PTE ADDRESS	System virtual address of the page table entry that describes the virtual page mapped into this physical page

(continued on next page)

Table SDA–18 (Cont.) Page Frame Number Information in the SHOW PFN_DATA Display

Item	Contents
BAK	Place to find context, as information about 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
REFCNT	Number of references being made to this page
FLINK	Address of the next page in the list in which this virtual page currently resides
BLINK	Address of the previous page in the list in which this virtual page currently resides
TYPE	Type of virtual page; one of the following: • 00 Process page • 01 System page • 02 Global, read-only page • 03 Global, read/write page • 04 Process page-table page • 05 Global page-table page

(continued on next page)

Table SDA–18 (Cont.) Page Frame Number Information in the SHOW PFN_DATA Display

Item	Contents
STATE	State of the virtual page, the low nibble of which can be one of the following: • 0 Page is on the free page list. • 1 Page is on the modified page list. • 2 Page is on the bad page list. • 3 Release of the page to the free or modified page list is pending. • 4 Error occurred as the page was being read from the disk. • 5 Modified page writer is currently writing the page to the disk. • 6 Page fault handler is currently reading the page from the disk. • 7 Page is active and valid.

Example

```
SDA>SHOW PFN_DATA

Free page list
--------------
Count:             225
Low limit:          57
High limit: 1073741824

PFN   PTE ADDRESS   BAK     REFCNT FLINK BLINK    TYPE        STATE
----  ----------- --------  ------ ----- -----  ----------  ----------
1329    8047AF3C  03002A83      0   1963 0000   00 PROCESS  00 FREELST
1963    8047AB10  03002A43      0   017C 1329   00 PROCESS  00 FREELST
017C    8047B3F8  03002A84      0   14B4 1963   00 PROCESS  00 FREELST
14B4    8047B464  03002A85      0   1529 017C   00 PROCESS  00 FREELST
1529    8047AA34  03002A87      0   1485 14B4   00 PROCESS  00 FREELST
1485    8047AC80  030010B3      0   1707 1529   00 PROCESS  00 FREELST
   .
   .
   .
```

In this example, the SHOW PFN_DATA command displays the information for the free page list, the modified page list, and the bad page list, and then all of the PFN database, including the first three lists.

SHOW POOL

Displays information about the disposition of paged and nonpaged memory, nonpaged dynamic storage pool, and paged dynamic storage pool.

Format

SHOW POOL [range][/ALL | /FREE | /HEADER | /NONPAGED | /PAGED | /RING_BUFFER | /STATISTICS | /SUMMARY | /TYPE=block-type]

Parameters

range
Range of virtual addresses in pool that SDA is to examine. You can express a range using the following format:

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 allocated pool, including the pool lists, nonpaged dynamic storage pool, and paged dynamic storage pool. This is the default behavior of the SHOW POOL command.

/FREE
Displays the entire contents, both allocated and free, of the specified region or regions of pool. You cannot use the /FREE qualifier when you use a **range** to indicate a region of pool to be displayed.

/HEADER
Displays only the first 16 longwords of each data block found within the specified region or regions of pool.

/NONPAGED
Displays the contents of the nonpaged dynamic storage pool currently in use.

/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, the most recent to the least recent. You cannot use this qualifier with any other SHOW POOL qualifier. This qualifier is most useful when analyzing crash dumps; output might not be consistent when used on a running system.

/STATISTICS
Displays usage statistics about each pool list if pool-checking has been enabled. For each list, the following are displayed:

- Queue header address
- Packet size

- Attempts, failures, and deallocations

  SDA does not synchronize its access to these last three counters with other CPUs in a symmetric multiprocessing (SMP) system. Therefore, the numbers might not add up to what you would expect in a multiprocessor configuration. However, the statistics do provide a good indicator of overall pool activity.

/SUMMARY
Displays only an allocation summary for each specified region of pool.

/TYPE=block-type
Displays the blocks within the specified region or regions of pool that are of the indicated **block-type**. If SDA finds no blocks of that type in the pool region, it displays a blank screen, followed by an allocation summary of the region.

Description

The SHOW POOL command displays information about the contents of any specified region of pool in an 8-column format. Following are explanations and examples of the contents of the full display.

- Column 1 contains the type of control block that starts at the virtual address in pool indicated in column 2. If SDA cannot interpret the block type, it displays a block type of "UNKNOWN." Column 3 lists the number of bytes (in decimal) of memory allocated to the block. The block size is fixed for SRPs, IRPs, and LRPs, and is variable in the paged and nonpaged pools. For example:

```
Col. 1  Col. 2     Col. 3
------  --------   ------
CIMSG   80BADE00   208
```

- The remaining columns contain a dump of the contents of the block, in 4-longword intervals, until the block is complete. Columns 4 through 7 display, from right to left, the contents in hexadecimal; column 8 displays, from left to right, the contents in ASCII. If the ASCII value of a byte is not a printing character, SDA displays a period (.) instead. For example:

```
Col. 4   Col. 5   Col. 6   Col. 7   Col. 8
-------- -------- -------- -------- ----------------
001000DA 003C0090 0000A900 00036FF0 .o........<.Ú...
D9B3001C 00000000 A0B5001D 35E60017 ...5.............
41414141 00000600 65EA0004 00000600 .......e....AAAA
41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
```

- For each region of pool it examines, the SHOW POOL command displays an allocation summary. This 4-column table lists, in column 2, the types of control blocks identified in the region and records the number of each in column 1. The last two columns represent the amount of the pool region occupied by each type of control block: column 3 records the total number of bytes, and column 4 records the percentage. The summary concludes with an indication of the number of bytes used within the particular pool region, as well as the number of bytes remaining. It provides an estimate of the percentage of the region that has been allocated. For example:

```
Col.1  Col. 2         Col.3   Col. 4
-----  -------        -----   ------
    3  UNKNOWN   =      176  (29%
    2  CIDG      =      288  (48%
    1  CIMSG     =      144  (24%)

Total space used = 608 out of 608 total bytes, 0 bytes left

Total space utilization = 100%
```

Examples

1.
```
SDA> SHOW POOL G0BADE00;260

Non-paged dynamic storage pool
------------------------------
                    Dump of blocks allocated from non-paged pool


CIMSG   80BADE00   144
                      001000DA 003C0090 0000A900 00036FF0 .o........<.....
                      D9B3001C 00000000 A0B5001D 35E60017 ...5............
                      41414141 00000600 65EA0004 00000600 .......e....AAAA
                      41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                      41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
   .
   .
   .
UNKNOWN 80BADE90   112
                      41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                      41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                      41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                      41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
   .
   .
   .
CIDG    80BADED0   144
                      807708BB 003B0090 0004D7E0 000008F0 .........;...w.
                      61616161 61616161 61616161 016CE87C ..l.aaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
   .
   .
   .
UNKNOWN 80BADF60    64
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
   .
   .
   .
CIDG    80BADFA0   144
                      807708BB 003B0090 0003FFC0 0004B1B0 .........;...w.
                      61616161 61616161 61616161 016CE94C L.l.aaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
   .
   .
   .
UNKNOWN 80BAE030    48
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                      61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
```

```
Summary of non-paged pool contents

    3  UNKNOWN   =      176  (29%)
    2  CIDG      =      288  (48%)
    1  CIMSG     =      144  (24%)

Total space used = 608 out of 608 total bytes, 0 bytes left

Total space utilization = 100%
```

This example, which uses a range of values, examines 608 (260_{16}) bytes of nonpaged pool, starting at address $80BADE00_{16}$. SDA attempts to identify allocated blocks as it proceeds through the specified region of pool, and displays an allocation summary when it completes the listing.

2.
```
SDA> SHOW POOL/FREE

Non-paged dynamic storage pool
------------------------------

   Dump of blocks allocated from non-paged pool

UNKNOWN 80E7C400 67136
                         0000E53B 80E9EC00 00010000 80F16625 %fñ......ìé.;å..
                         0000E53B 80E9EC00 00010001 80F16625 %fñ......ìé.;å..
                         0000E53B 80E9EC00 00010000 80F166A3 £fñ......ìé.;å..
                         0000E53B 80E9EC00 00010001 80F166A3 £fñ......ìé.;å..
                         0000E53B 80E9EC00 00010000 80F16066 f........ìé.;å..
                         0000E53B 80E9EC00 00010001 80F16066 f........ìé.;å..
                         0000E53B 80E9EC00 00010000 80F16F32 2oñ......ìé.;å..
                         0000E53B 80E9EC00 00010001 80F16F32 2oñ......ìé.;å..
                         0000E53D 80EA1B08 00010000 80F16F48 Hoñ.......ê.=å..
                         0000E53D 80E9EC00 00010001 80F16F48 Hoñ......ìé.=å..
                         0000E53D 80E9EC00 00010000 80F170D8 Øpñ......ìé.=å..
    .
    .
    .
```

The SHOW POOL/FREE command in this example produces a display similar in format and extent to that presented in Example 1. However, it displays the unallocated portions of pool in addition to those that are used.

3. SDA> SHOW POOL/PAGED/HEADER

```
Paged dynamic storage pool
--------------------------
                    Dump of blocks allocated from paged pool


RSHT    8024FE00   528
                     802DC710 00380210 00000000 FFFFFF80 ..........8...-.
LNM     80250010    96
                     8015B847 00400060 802D75A0 00000000 .....u-.‘.@.G...
LNM     80250070    48
                     8015B847 01400030 802500A0 802D7400 .t-...%.0.@.G...
LNM     802500A0    96
                     8015B847 02400060 802DC170 80250070 p.%.p.-.‘.@.G...
LNM     80250100    48
                     8015B847 00400030 802DC510 802E1B60 ‘.....-.0.@.G...
   .
   .
   .
```

The SHOW POOL/PAGED/HEADER command displays only the name of each block allocated from paged pool, its starting address, its size, and the first four longwords of its contents.

4. SDA SHOW POOL/RING_BUFFER

```
(Non-Paged Pool History Ring-Buffer
 (512 entries: Most recent first)
Packet Adr  Size  Type  Subtype Caller’s PC   Routine called      Entry Adr
 --------  -----  -----  ----    --------    ---------------      --------
 DA9EE5C0    168  IRP      3     D8012BF1    EXE$DEANONPAGED      DA4C7750
 DAA27EC0    192  DSRV     3     DA591941    EXE$DEANONPAGED      DA4C7740
 DAD47B40    168  IRP      0     DA591918    EXE$DEANONPAGED      DA4C7730
 DAAB5400     24  FRK     52     DA590252    EXE$DEANONPAGED      DA4C7720
 DAAB5400     24  TQE      0     DA591276    EXE$ALONONPAGED      DA4C7710
 DAD47B40    168  IRP     64     DA59184A    EXE$ALONONPAGED      DA4C7700
 .
 .
 .
 DAA66500    172  IRP     64     DB251C80    EXE$ALONONPAGED      DA4C7770
 DAA32300    192  CIMSG    0     DA54C2C8    EXE$DEANONPAGED      DA4C7760
```

This example of the SHOW POOL/RING_BUFFER command displays the contents of the nonpaged pool history ring buffer, with the most recent entries displayed first.

5. SDA SHOW POOL/STATISTICS

```
        List head    List      Alloc.       Alloc.
         Address     Size     Attempts     Failures      Deallocs.
        --------    -----    ----------   ----------    ----------
        D80A9030       64       2077039         1121       2073964
        D80A9038      128       6323789         4502       6309357
        D80A9040      192      21085351         1903      21078538
        D80A9048      256        502388         2025        499705
        D80A9050      320       1372168         3512       1367707
        D80A9058      384         32649          774         31899
        D80A9060      448       2463316         1025       2462243
        D80A9068      512        357170         2181        354754
        D80A9070      576        293998         2438        291476
        D80A9078      640        168145          645        167482
        D80A9080      704         83645         2043         81547
        D80A9088      768         34852          120         34726
        D80A9090      832         21263           44         21215
        .
        .
        .
        D80A9290     4928       2305645         3283       2302249
        D80A9298     4992             9            0             6
        D80A92A0     5056             0            0             0
        D80A92A8     5120             1            0             0
```

This example of the SHOW POOL/STATISTICS command displays usage statistics about each pool list.

6. SDA SHOW POOL/SUMMARY

```
Summary of non-paged pool contents

  145  UNKNOWN   =   191616  (18%)
    2  ADP       =     1280  (0%)
   35  ACB       =     2624  (0%)
    3  AQB       =      192  (0%)
   17  CRB       =     2368  (0%)
   16  DDB       =     2048  (0%)
  355  FCB       =   113600  (11%)
    3  FRK       =    18240  (1%)
   16  IDB       =     1088  (0%)
   42  IRP       =     8064  (0%)
   20  PCB       =    10240  (1%)
   48  TQE       =     3072  (0%)
   70  UCB       =    21696  (2%)
    5  VCB       =     1280  (0%)
  299  WCB       =    51008  (5%)
  287  BUFIO     =   112128  (11%)
    5  TYPAHD    =     1920  (0%)
    2  MVL       =     4736  (0%)
    3  NET       =     4160  (0%)
   15  CXB       =    23616  (2%)
    5  NDB       =     2112  (0%)
   14  DPT       =   132928  (13%)
    .
    .
    .
Total space used = 1016896 out of 1068032 total bytes, 51136 bytes left
Total space utilization = 95%
Summary of paged pool contents
```

```
   33  UNKNOWN   =    36480  (15%)
    1  PQB       =     2256  (0%)
  224  GSD       =    14240  (6%)
  153  KFE       =    10864  (4%)
    3  MTL       =       96  (0%)
  118  KFRH      =    46736  (20%)
    1  RSHT      =      528  (0%)
    1  XWB       =    18048  (7%)
  225  LNM       =    16720  (7%)
    4  KFD       =      224  (0%)
    1  KFPB      =       16  (0%)
    2  CIA       =    29264  (12%)
    1  CHIP      =     9216  (4%)
   41  ORB       =     5248  (2%)
    2  ARB       =    34912  (15%)
    1  PTC       =     3072  (1%)
    7  OCB       =     1344  (0%)
    1  PGD       =      208  (0%)

Total space used = 229472 out of 524800 total bytes, 295328 bytes left

Total space utilization = 43%
```

This example of the SHOW POOL/SUMMARY command displays an allocation summary for each region of pool.

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).[10]

/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=name
Displays virtual circuit (VC) information associated with the named node on the specified PDT. You must use this qualifier with /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 SCS connections. This information is used by all system communications services (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

[10] 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. CDT addresses are also stored in many individual data structures related to SCS connections; for instance, in the path block displays of the SHOW CLUSTER/SCS command.

- 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:

Symbol	Explanation or Example
VC_nodename	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	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	MGMT_VCRP_ETA, address of the management VCRP for BUS ETA
HELLO_VCRP_busname	HELLO_VCRP_ETA, address of the HELLO message VCRP for BUS ETA
VCIB_busname	VCIB_ETA, address of the VCIB for BUS ETA
UCB_LAVC_busname	UCB_LAVC_ETA, address of the LAN device's UCB used for the local area VAXcluster protocol
UCB0_LAVC_busname	UCB0_LAVC_ETA, address of the LAN device's template UCB
LDC_LAVC_busname	LDC_LAVC_ETA, address of the LDC structure associated with LAN device ETA
LSB_LAVC_busname	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, after the equal sign in SDA commands.

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 of 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.

Examples

1. SDA> SHOW PORTS/ADDR=PE_PDT

```
VAXcluster data structures
--------------------------

                  --- Port Descriptor Table (PDT) 806C37A0 ---

Type: 03 pe
Characteristics: 0000

Msg Header Size           32  Connect        80799F94  Recyclh_Msg_Buf  8079AD8A
Max Xfer Bcnt       FFFFFFFF  Dealloc_Dg_Buf 8079AFDA  Request_Data     8079B1CC
DG Header Size           288  Disconnect     8079A06B  Send_Data        8079B215
Poller Sweep              31  Unmap          8079B510  Send_Dg_Buf      8079B03E
Fork Block W.Q.        empty  Map            8079B111  Send_Msg_Buf     8079AEA8
UCB Address         806C0E50  Map_Bypass     8079B0F8  Send_Cnt_Msg_Buf 8079AEAF
ADP Address         00000000  Map_Irp        8079B101  Read_Count       80796D59
Accept              80799FEC  Map_Irp_Bypass 8079B0F0  Rls_Read_Count   80796DD3
Alloc_Dg_Buf        8079AFC6  Queue_Dg_Buf   8079AFE0  Mreset           80799C94
Alloc_Msg_Buf       8079AD05  Queue_Mult_Dgs 8079AFE8  Mstart           80799C9E
Dealloc_Msg_Buf     8079ADE3  Recycl_Msg_Buf 8079AD94  Stop_Vcs         8079BEDD
Dealloc_Msg_Buf_Reg 8079ADF6  Reject         8079A036  Send_Dg_Reg      8079B031

                 --- Port Block 80B091B0 ---

Status: 0001 authorize
VC Count: 5
Secs Since Last Zeroed: 311728

SBUF Size                436             LBUF Size              1788
SBUF Count                12             LBUF Count                1
SBUF Max                 768             LBUF Max                384
SBUF Quo                  13             LBUF Quo                  1
SBUF Miss                 18             LBUF Miss             12235
SBUF Allocs           499579             LBUF Allocs           16824
SBUFs In Use               0             LBUFs In Use              0
Peak SBUF In Use          14             Peak LBUF In Use         34
SBUF Queue Empty           0             LBUF Queue Empty          0
TR SBUF Queue Empty        0
No SBUF for ACK            0

Bus Addr  Bus     LAN Address    Error Count Last Error   Time of Last Error
--------  ---  ----------------- ----------- ---------- ----------------------
80B08920  LCL  00-00-00-00-00-00           0
80B08090  ESA  AA-00-04-00-33-FD          75  00000334  25-MAR-1993 23:39:28.27
80B008B0  XQA  08-00-2B-0A-6A-6B          12  0000002C  23-MAR-1993 12:43:59.07
80AF6E90  XQB  08-00-2B-08-CB-B8           0

                 --- Virtual Circuit (VC) Summary ---

VC Addr     Node    SCS ID  Lcl ID    Status Summary        Last Event Time
--------  --------  ------  ------  -----------------   ----------------------
806CD1A0  NODE12    64819  223/DF  open,path           1-JAN-1993 00:00:00.03
806CD6E0  NODE13    64856  222/DE  open,path           1-JAN-1993 00:00:07.
806CD9A0  NODE14    64587  221/DD  open,path           22-MAR-1993 18:34:10.18
8070D530  NODE15    64555  220/DC  open,path           22-MAR-1993 18:57:33.
8074AB60  NODE16    64841  219/DB  open,path           25-MAR-1993 20:42:38.20
```

The SHOW PORTS/ADDRESS command displays the port descriptor table (PDT) structure, some of the fields in the PORT structure, the BUS summary, and the virtual circuit summary.

2. SDA>SHOW PORTS/BUS=BUS_ESA

```
VAXcluster data structures
--------------------------
--- BUS: 80B08090  (ESA)  Device: ES_LANCE  LAN Address: AA-00-04-00-33-FD---
                           LAN Hardware Address: 08-00-2B-12-AE-A1
Status: 00000A03 run,online,xmt_chaining_disabled,restart
------- Transmit ------  ------- Receive -------  ---- Structure Addresses ---
Msg Xmt          434107  Msg Rcv         1170090  PORT Address       80B091B0
  Mcast Msgs     103939    Mcast Msgs     859601  VCIB Addr          80B08248
  Mcast Bytes  13304192    Mcast Bytes  96272072  HELLO Message Addr  80B082D8
Bytes Xmt      59789962  Bytes Rcv     146674695  BYE Message Addr    80B08468
Outstand I/Os         0  Buffer Size        1424  Delete BUS Rtn Adr  8079E424
Xmt Errors           75  Rcv Ring Size         8
Last Xmt Error 00000334         Time of Last Xmt Error 25-MAR-1993 23:39:28.27
--- Receive Errors ----  ------ BUS Timer ------  ----- Datalink Events ------
TR Mcast Rcv          0  Handshake TMO  8079FA50  Last 22-MAR-1993 18:25:25.12
Rcv Bad SCSID         0  Listen TMO     8079FA54  Last Event          00001202
Rcv Short Msg         0  HELLO timer           1  Port Usable                1
Fail CH Alloc         0  HELLO Xmt err        38  Port Unusable              0
Fail VC Alloc         0                           Address Change             1
Wrong PORT            0                           Port Restart Fail          0
```

The SHOW PORTS/BUS=BUS_id command displays the data for the specified BUS structure. The last event time is at the top of the lower right-hand column. If an error was counted, the last error time is displayed under Xmt Errors. The normal status is: RUN, ONLINE, and RESTART.

The Xmt Error field indicates a problem detected during transmission of a message. The error rate should be less than one per hour.

3. SDA> SHOW PORTS/VC=VC_BREE

```
VAXcluster data structures
--------------------------
                 --- Virtual Circuit (VC) 806CD6E0 ---
Remote System Name:  BREE   (0:VAX)     Remote SCSSYSTEMID:  64856
Local System ID:  222 (DE)              Status: 0005 open,path
------ Transmit -------  ------ VC Closures ----  ---- Congestion Control ----
Msg Xmt          216686  SeqMsg TMO            0  UnAcked Msgs               1
  Unsequence          3  CC DFQ Empty          0  Pipe Quota Reached        33
  Sequence       149643  Topology Change       0  CMD Queue Len              0
  ReXmt             545  NPAGEDYN Low          0  Max CMD Queue Len          5
  Lone ACK        66495                           RSVP Threshold            15
Bytes Xmt      33309074                           Pipe Quota                31
------- Receive -------  - Messages  Discarded -  ----- Channel Selection ----
Msg Rcv          194492  No Xmt Chan           0  Preferred Channel   80704320
  Unsequence          1  Rcv Short Msg         0  Delay Time          FB7E6F80
  Sequence       178905  Illegal Seq Msg       0  Buffer Size             1424
  ReRcv              30  Bad Checksum          0  Channel Count              6
  Lone ACK        15531  TR DFQ Empty          0  Channel Selections      3920
  Cache              26  TR MFQ Empty          0  Protocol               1.3.0
  Ill ACK             0  CC MFQ Empty          0  Open  1-JAN-1993 00:00:07.03
Bytes Rcv      52086897  Cache Miss            0  Cls  17-NOV-1858 00:00:00.00

        -- Channel Summary for Virtual Circuit (BREE  ) 806CD6E0 --

Address     Type    Xmt Time Size Preferred    Best        Last State Change
--------  --------- -------- ---- ---------  --------   ----------------------
80704320  Preferred FB7E6F80 1424       812       617   22-MAR-1993 18:14:07.01
807043E0  Active    FB7E735E 1424        95         4   25-MAR-1993 20:01:15.18
807050D0  Active    FB7E7FED 1424       431         0   25-MAR-1993 20:01:15.18
806CD820  Active    FB7E728E 1424       868      1470   25-MAR-1993 20:01:15.18
80705010  Active    FB7E7043 1424       738         9   25-MAR-1993 20:00:58.17
806CD8E0  Active    FB7E7BB5 1424       976      1744   25-MAR-1993 20:00:31.17
```

The SHOW PORTS/VC=VC_id command displays the virtual circuit data for the specified remote node and a channel summary. In this display, the upper center of the display contains the virtual circuit status. The lower right-hand corner contains the virtual circuit open and close times.

The ReXmt field indicates a problem sending messages to the remote system. The error rate per hour should be less than the Pipe Quota field.

The ReRcv field indicates a problem receiving messages from the remote system. The error rate per hour should be less than the Pipe Quota field.

4. `SDA> SHOW PORTS/MESSAGE/VC=address`

This SHOW PORTS command displays the virtual circuit data for the specified remote node, followed by the message data for the remote node. The virtual circuit message display shows the counters for the following items:

– Sequenced message delivery

– Any messages in the process of being transmitted or in the receive cache

The following is an example of part of a display resulting from the SHOW PORTS/MESSAGE/VC=vc-address command:

```
VAXcluster data structures
--------------------------

        --- Sequenced Message Counters Virtual Circuit (VC) 806CD6E0 ---

      NSU: 4457   HAA: 4456   LAR: 4455   HSR: B3AA  Cache Mask: 00000000

                           Messages Waiting for ACKs

VCRP adr  Len  Flgs Seq   Ack                    Message Data
--------  ---- ---- ----  ----  ------------------------------------------------
806CD2E0   137  0B  4456  B3AA  02 7D 00 04 00 0A 00 00 00 09 00 D 75 05 00 67
```

5. `SDA> SHOW PORTS/CHANNEL=CH_BREE`

This SHOW PORTS command displays the data for the specified channel. The normal state is OPEN, with a status of PATH, OPEN, and RMT_HWA_ VALID.

In the following example display resulting from this command, the top of the display shows the remote device name, the remote device type, and the channel open and close times.

```
VAXcluster data structures
--------------------------

: PEDRIVER Channel (CH:80704320) for Virtual Circuit (VC:806CD6E0) BREE   --
State: 0004 open               Status: 0B path,open,rmt_hwa_valid
BUS: 80B008B0  (XQA)  Lcl Device: XQ_DELQA  Lcl LAN Address: 08-00-2B-0A-6A-6B
Rmt Name: XQB         Rmt Device: XQ_DEQTA  Rmt LAN Address: 08-00-2B-13-70-88
Rmt Seq #: 0002   Open:22-MAR-1993 18:14:07.01  Closed:17-NOV-1858 00:00:00.00
------- Transmit ------  ------- Receive -------  ----- Channel Selection ----
Lcl CH Seq #       0001  Msg Rcv          139205  Average Xmt Time    FB879740
Msg Xmt           66707    Mcast Msgs     103906  Remote Buffer Size      1424
  Ctrl Msgs           1    Mcast Bytes  10182788  Max Buffer Size         1424
  Ctrl Bytes         98    Ctrl Msgs           2  Best Channel             615
Bytes Xmt       9130385    Ctrl Bytes        196  Preferred Channel        810
Rmt Ring Size        31  Bytes Rcv      22654333  Retransmit Penalty         2
--------------  Channel Errors  --------------  Xmt Error Penalty         12
Handshake TMO         0  Short CC Msgs         0  ------- Channel Timer ------
Listen TMO            0  Incompat Chan         0  Timer Entry Flink   8079FA3C
Bad Authorize         0  No MSCP Srvr          0              Blink   80705010
Bad ECO               0  Disk Not Srvd         0  Last Ring Index           08
Bad Multicast         0  Old TR Msgs           0  Protocol               1.3.0
Topology Change       0                           Supported Services  00000000
```

6. SDA> SHOW PORTS/DEVICE/CHANNEL/VC=vc-address

   This SHOW PORTS command displays the following information:

   – Virtual circuit data for the specified remote node

   – Channel data

   – The network path description for each channel to the remote node

   The following is an example of a display resulting from the SHOW PORTS/DEVICE/CHANNEL/VC=vc-address command:

```
VAXcluster data structures
--------------------------
: Network Component List (CLST:80D36440) for Channel (CH:806DC420) --

COMP adr   COMP Type  Description
--------   ---------  -----------------------------------------------------
80D30010   NODE       SGRPOP:VAXstation 3300; RDO3-4/U10
80CC9300   ADAPTER    ESA; SGRPOP:VAXstation 3300; RDO3-4/U10 (08-00-2B-12-AE-A1)
80D3CDB0   COMPONENT  RD34C4, I-Cluster Segment DAMPR
80D40380   COMPONENT  RD34C4, I-Cluster Segment SELNI
80D36AD0   COMPONENT  I-Cluster Segment
80D2D4C0 P COMPONENT  RDO3-4 Lab, DIVER: I-Cluster Segment SELNI
   .
   .
   .
80D323F0   NODE       PELLNM:rack mounted MicroVAX II; RDO3-4 Lab
```

   This display is useful after the local area VAXcluster network failure analysis data has been loaded. After a network failure analysis, this display indicates primary and secondary failed component suspects in the following ways:

   - P: Primary suspect
   - S: Secondary suspect
   - ?: Component that cannot be proved to be working

7. SDA> SHOW PORTS /DEVICE /CHANNEL=address

   This SHOW PORTS command displays the channel data and the network path description if it was provided by the network failure analysis.

8. SDA> SHOW PORTS/BUS/CHANNEL/DEVICE/MESSAGE/VC/ADDRESS=PE_PDT

   This command displays all of the bus structures, all of the virtual circuits and their message counters, and channels, including network path descriptions when available.

9. SDA> SHOW PORTS/ADDR=862C8D80/NAME=DAVID3

```
VAXcluster data structures
--------------------------


   --- Virtual Circuit (VC) 862C8D80 ---
Remote System Name:  DAVID3 (0:VAX)     Remote SCSSYSTEMID:  64588
Local System ID:  213 (D5)              Status: 0005 open,path
------ Transmit -------  ------ VC Closures ----  ---- Congestion Control ----
Msg Xmt              19  SeqMsg TMO            0  Pipe Quota/Slo/Max   1/31/31
  Unsequence         16  CC DFQ Empty          0  Pipe Quota Reached         0
  Sequence            3  Topology Change       0  Xmt C/T                  0/1
  ReXmt             0/0  NPAGEDYN Low          0  RndTrp uS          3000000+0
  Lone ACK            0                          UnAcked Msgs               0
Bytes Xmt          1058                          CMD Queue Len/Max        0/0
------- Receive -------  - Messages  Discarded -  ----- Channel Selection ----
Msg Rcv              10  No Xmt Chan           0  Preferred Channel   00000000
  Unsequence         16  Rcv Short Msg         0  Delay Time          003266DB
  Sequence            0  Illegal Seq Msg       0  Buffer Size             1424
  ReRcv               0  Bad Checksum          0  Channel Count              2
  Lone ACK            0  TR DFQ Empty          0  Channel Selections         9
  Cache               0  TR MFQ Empty          0  Protocol                1.3.0
  Ill ACK             0  CC MFQ Empty          0  Open  8-FEB-1993 11:30:43.60
Bytes Rcv           440  Cache Miss            0  Cls   8-FEB-1993 11:28:30.69

 -- Channel Summary for Virtual Circuit (DAVID3) 862C8D80 --

Address     Type    Xmt Time Size Preferred    Best       Last State Change
--------  --------- -------- ---- ---------  --------  ----------------------
862CB600  Active    000927BF 1424         3         4  8-FEB-1993 11:30:53.69
862C8F00  Active    000927BF 1424         6         2  8-FEB-1993 11:30:43.60
```

   The command in this example displays virtual connect information associated with the DAVID3 node, which is associated with the port descriptor table whose address is 862C8D80.

SHOW PROCESS

Displays the software and hardware context of any process in the balance set.

Format

SHOW PROCESS [/qualifier[,...]][ALL | process-name | /INDEX=nn | /SYSTEM]

Parameters

ALL
Shows information about all processes that exist in the system.

process-name
Name of the process for which information is to be displayed.[11]

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 ($) characters. If it contains any other characters, you must enclose the **process-name** in quotation marks (" ").

Qualifiers

/ALL
Displays all information shown by the following qualifiers: /CHANNEL, /PAGE_TABLES, /PCB, /PHD, /PROCESS_SECTION_TABLE, /REGISTERS, and /WORKING_SET.

/CHANNEL
Displays information about the I/O channels assigned to the process.

/IMAGES
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.

/INDEX=nn *or* /ID=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). You can supply either of the following values for **nn**:

- The process index itself
- The process identification (PID) or extended PID longword, from which SDA extracts the correct index

To obtain these values for any given process, issue the SDA command SHOW SUMMARY.

[11] 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 4 for information about how this can affect the process context—and CPU context—in which SDA commands execute.)

/LOCKS
Displays the lock management locks owned by the current process.

The /LOCKS qualifier produces a display similar in format to that produced by the SHOW LOCKS command. See Table SDA–15 for additional information.

/P0
Displays the page tables for P0 space. See the description of the /PAGE_TABLES qualifier.

/P1
Displays the page tables for P1 space. See the description of the /PAGE_TABLES qualifier.

/PAGE_TABLES*or* /PPT [range|/P0|/P1]
Displays the page tables P0 and P1 spaces, or, optionally, either the page table or the page table entries for a **range** of addresses.

You can express a **range** using the following format:

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* pages, starting with page *m*

/PARTICIPANTS[=DISPLAY=(item [,...])]
Displays information about all transactions for the process. The argument to DISPLAY can be either a single item or a list. The following items can be specified.

Item	Description
ALL	All transaction control structures for the transactions. This is the default behavior.
BRANCHES	Control structures for branches of the transactions.
PARTICIPANTS	Control structures for resource managers participating in the transactions.
THREADS	Control structures for threads of the transactions.
TRANSACTIONS	Transaction control structures for the transactions.

/PCB
Displays the information contained in the software process control block (PCB). This is the default behavior of the SHOW PROCESS command.

/PHD
Lists information included in the process header (PHD).

/PROCESS_SECTION_TABLE *or* /PST
Lists the information contained in the process section table (PST).

/REGISTERS
Lists the hardware context of the process, as reflected in the registers of the process stored in the hardware PCB and—if the process is current on a processor in the system—the registers of the processor.

/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 SDA–14 for a list of keywords that you can supply 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.

/SYSTEM

Displays the system process control block.[12] The system PCB and process header (PHD) are dummy structures that are located in system space. These structures contain the system working set, global section table, global page table, and other systemwide data.

/TRANSACTIONS=(option[,...])

Displays information about all transactions, or the specified transaction, for the process. The following two options can be specified either together or separately:

- DISPLAY=(item [,...])

  Specifies the type of information to be displayed. The argument to DISPLAY can be either a single item or a list. The following items can be specified.

Item	Description
ALL	All transaction control structures for the specified transaction. This is the default behavior.
BRANCHES	Control structures for branches of the specified transaction.
PARTICIPANTS	Control structures for resource managers participating in the specified transaction.
THREADS	Control structures for threads of the specified transaction.
TRANSACTIONS	Transaction control structures for the specified transaction.

[12] 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 4 for information about how this can affect the process context—and CPU context—in which SDA commands execute.)

- TID=tid

  Specifies the transaction for which information is to be displayed. If you omit the TID option, the SHOW PROCESS/TRANSACTIONS command displays information about all transactions for the process.

If you omit these options, the SHOW PROCESS/TRANSACTIONS command displays all information about all transactions for the process.

Note that the SHOW PROCESS/TRANSACTIONS and SHOW PROCESS/PARTICIPANTS commands display the same information about transactions, but in different orders. The SHOW PROCESS/TRANSACTIONS command walks down a transaction queue. The SHOW PROCESS/PARTICIPANTS command walks down a resource manager queue.

/VECTOR_REGS
Displays the saved process vector registers.

/WORKING_SET *or* /WSL
Displays the working set list of the process.

Description

The SHOW PROCESS command displays information about the process specified by **process-name**, the process specified with the /INDEX qualifier, the system process, or all processes. By default, the SHOW PROCESS command produces information about the SDA current process, as explained in Section 4.

The SHOW PROCESS command performs an implicit SET PROCESS command under certain uses of its qualifiers and parameters, as explained in Section 4, Section 5, and Section 6. If you use the SHOW PROCESS command and name a process that is the current process on a CPU, SDA temporarily assigns the symbols shown in Table SDA–9 to the values in the process. You can then refer to those symbols when you use the FORMAT command.

The default of the SHOW PROCESS command provides information taken from the software process control block (PCB).[13] This information describes the following characteristics of the process:

- Software context
- Condition-handling information
- Information about interprocess communication
- Information about counts, quotas, and resource usage

Among the displayed information are the PID, EPID, priority, job information block (JIB) address, and process header (PHD) address of the process. SHOW PROCESS also describes the resources owned by the process, such as event flags and mutexes. The “State” field records the current scheduling state of the process; in a multiprocessing system, the display indicates the CPU ID of any process whose state is CUR.

The SHOW PROCESS/ALL command displays additional process-specific information, also provided by several of the individual qualifiers to the command.

[13] This is the first display provided by the /ALL qualifier and the only display provided by the /PCB qualifier.

The **process header** display, also produced by the /PHD qualifier, provides information taken from the process header (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's use of the following resources:

- Process memory
- The pager
- The scheduler
- Asynchronous system traps
- I/O activity
- CPU activity

The **process registers** display, also produced by the /REGISTERS qualifier, describes the hardware context of the context, as reflected in its registers.

The hardware context of a process is stored in two places:

- If the process is currently executing on a processor in the system (that is, in the CUR scheduling state), its hardware context is contained in that processor's registers. (That is, the registers of the process and the registers of the processor 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 hardware context is stored in the part of the PHD known as the hardware PCB.

The **process registers** display first lists those registers stored in the hardware PCB ("Saved process registers"). If the process to be displayed is currently executing on a processor in the 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:

- General-purpose registers (R0 through R11 and the AP, FP, and PC)
- Stack pointers (KSP, ESP, SSP, and USP)
- Special-purpose registers (PC and PSL)
- Base and length registers (P0BR, P1BR, P0LR, and P1LR)

The **working set information** and **working set list** displays, also produced by the /WORKING_SET 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 the following information for each entry in the working set list.

Column	Contents
INDEX	Index into the working set list at which information for this entry can be found
ADDRESS	Virtual address of the page in the process address space that this entry describes

Column	Contents
STATUS	Three columns that list the following status information: • Page type • Location of the page in physical memory • Indication of whether the page is locked into the working set

When SDA locates one or more unused working set entries, it issues the following message:

```
--- n empty entries
```

In this message, *n* is the number (in decimal) of contiguous, unused entries.

The **process section table information** and **process section table** displays, also produced by the /PROCESS_SECTION_TABLE qualifier, list each entry in the process section table (PST) and display the offsets to the first free entry and last used entry.

SDA displays the information listed in Table SDA–19 for each PST entry.

Table SDA–19 Process Section Table Entry Information in the SHOW PROCESS Display

Part	Definition
INDEX	Offset into the PST at which the entry is found. Because entries in the process section table begin at the highest location in the table, and the table expands toward lower addresses, the following expression determines the address of an entry in the table: PHD + PSTBASOFF—INDEX.
ADDRESS	Virtual address that marks the beginning of the first page of the section described by this entry.
PAGES	Length, in pages, of the process section.
VBN	Virtual block number, the number of the file's virtual block that is mapped into the section's first page.
CLUSTER	Cluster size used when faulting pages into this process section.
REFCNT	Number of pages of this section that are currently mapped.
FLINK	Forward link, the pointer to the next entry in the PST list.
BLINK	Backward link, the pointer to the previous entry in the PST list.
FLAGS	Flags that describe the access that processes have to the process section.

The **P0 page table** and **P1 page table** displays, also produced by the /PAGE_TABLES qualifier, display listings of the page table entries of the process in the same format as that produced by the SHOW PAGE_TABLE command (see Tables SDA–16 and SDA–17).

The **process active channels** display, the last produced by SHOW PROCESS/ALL and the only one produced by the /CHANNEL qualifier, displays the following information for each I/O channel assigned to the process.

Column	Contents
Channel	Number of the channel
Window	Address of the window control block (WCB) for the file if the device is a file-oriented device; zero otherwise
Status	Status of the device: "Busy" if the device has an I/O operation outstanding; blank otherwise
Device/file accessed	Name of the device and, if applicable, name of the file being accessed on that device

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 SDA–20.

Table SDA–20 Process I/O Channel Information in the SHOW PROCESS Display

Information Displayed[1]	Type of Process
dcuu:	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.
dcuu:*filespec*	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*.
dcuu:(*file-id*)*filespec*	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.
dcuu:(*file-id*)	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*.

[1]This table uses the following formulas 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.

Examples

1. SDA> SHOW PROCESS

```
Process index: 001B   Name: PUTP1   Extended PID: 27E0011B
------------------------------------------------------------
Process status:  00044001   RES,BATCH,PHDRES

PCB address              803C7710    JIB address             806B9100
PHD address              81F5C400    Swapfile disk address   02002FA1
Master internal PID      0001001B    Subprocess count              0
Internal PID             0001001B    Creator internal PID    00000000
Extended PID             27E0011B    Creator extended PID    00000000
State                     CUR  00    Termination mailbox         0000
Current priority                3    AST's enabled               KES
Base priority                   3    AST's active                E
UIC                [00011,000176]    AST's remaining              39
Mutex count                     0    Buffered I/O count/limit     12/12
Waiting EF cluster              0    Direct I/O count/limit       18/18
Starting wait time       1B001C1C    BUFIO byte count/limit    31968/31968
Event flag wait mask     BFFFFFFF    # open files allowed left    90
Local EF cluster 0       20000001    Timer entries allowed left    9
Local EF cluster 1       C0000000    Active page table count       0
Global cluster 2 pointer 00000000    Process WS page count      1020
Global cluster 3 pointer 00000000    Global WS page count       233
```

The SHOW PROCESS command displays information taken from the software PCB of PUTP1, the SDA current process. According to the "State" field in the display, process PUTP1 is current on CPU 00 in the multiprocessing system.

2. SDA> SHOW PROCESS/ALL

```
Process index: 00AD   Name: GLOBE   Extended PID: 462002AD
------------------------------------------------------------

Process status:  02040001   RES,PHDRES

PCB address              8044E650    JIB address             806E0010
   .
   .
   .
Process header
--------------

First free P0 address      0007D600    Accumulated CPU time       00000559
Free PTEs between P0/P1      276902    CPU since last quantum         FFEE
First free P1 address      7FEF2200    Subprocess quota                 8
Free page file pages          24234    AST limit                       50
Page fault cluster size          16    Process header index          0020
Page table cluster size           2    Backup address vector      00003E12
Flags                          0002    WSL index save area        00003980
Direct I/O count                509    PTs having locked WSLs           5
Buffered I/O count              827    PTs having valid WSLs           20
Limit on CPU time          00000000    Active page tables              21
Maximum page file count       25600    Maximum active PTs              26
Total page faults              7589    Guaranteed fluid WS pages       20
File limit                       50    Extra dynamic WS entries       698
Timer queue limit                10    Locked WSLE counts array      1CD8
Paging file index          06000000    Valid WSLE counts array       2564

Saved process registers
-----------------------
R0   = 00000001    R1   = 00000000    R2   = 8000CA78    R3   = 8044E6A0
R4   = 8044E650    R5   = 00000000    R6   = 00000000    R7   = 00000003
R8   = 00001F60    R9   = 7FF9FB38    R10  = 7FF9FA08    R11  = 7FFE0070
```

```
AP   = 7FEF4AE4    FP   = 7FEF4AEC    PC   = 801622B4    PSL  = 03C00000
KSP  = 7FFE7E00    ESP  = 7FFE9E00    SSP  = 7FFED04E    USP  = 7FEF4AE4

P0BR = 82D43600    P0LR = 000003EB    P1BR = 82654E00    P1LR = 001FF792

Active registers for current process
------------------------------------
R0   = 00000001    R1   = 80002398    R2   = 00000000    R3   = 00000000
R4   = 7FFA05A0    R5   = 00000000    R6   = 0007D400    R7   = 00000010
R8   = 00001F60    R9   = 7FF9FB38    R10  = 7FF9FA08    R11  = 7FFE0070
AP   = 7FFE9D70    FP   = 7FFE9D58    PC   = 801620A5    PSL  = 01400000
KSP  = 7FFE7E00    ESP  = 7FFE9D58    SSP  = 7FFED04E    USP  = 7FEF4AE4

Working set information
-----------------------

First WSL entry          0074       Current authorized working set size   2048
First locked entry       00A6       Default (initial) working set size    512
First dynamic entry      00B9       Maximum working set allowed (quota)   2048
Last entry replaced      018C
Last entry in list       0561

Working set list
----------------

    INDEX   ADDRESS    STATUS

     0074  7FFE7C00     VALID PROCESS WSLOCK
     0075  7FFE7A00     VALID PROCESS WSLOCK
     0076  7FFE7800     VALID PROCESS WSLOCK
   .
   .
   .

Process section table information
---------------------------------

    Last entry allocated     FFA0
    First free PST entry     0000

Process section table
---------------------

INDEX  ADDRESS PAGES     WINDOW     VBN    CLUSTER  CHANNEL  REFCNT FLINK BLINK   FLAGS

 FFF8 00000200 0000000A  8082C400  00000002      0  7FFCCFD0    10    FFE8  FFF0
 FFF0 00001600 00000007  8082C400  0000000C      0  7FFCCFD0     0    FFF8  FFE8   WRT CRF
 FFE8 00002400 00000012  8082C400  00000013      0  7FFCCFD0    18    FFF0  FFF8
   .
   .
   .
P0 page table
-------------

ADDRESS  SVAPTE   PTE       TYPE  PROT BITS PAGTYP     LOC STATE TYPE REFCNT   BAK       SVAPTE FLINK BLINK

-------- 1 NULL PAGE

00000200 82D43604 F9804F73  VALID UR      U PROCESS ACTIVE  07   00     1   0040FFF8   82D43604 0000  0153
00000400 82D43608 F9806905  VALID UR      U PROCESS ACTIVE  07   00     1   0040FFF8   82D43608 0000  0154
00000600 82D4360C F9807569  VALID UR      U PROCESS ACTIVE  07   00     1   0040FFF8   82D4360C 0000  0155
   .
   .
   .
P1 page table
-------------

ADDRESS  SVAPTE   PTE       TYPE  PROT BITS PAGTYP     LOC STATE TYPE REFCNT   BAK       SVAPTE  FLINK BLINK

7FEF2400 82E52C48 21800000  DZERO UW      U
7FEF2600 82E52C4C 21800000  DZERO UW      U
7FEF2800 82E52C50 21800000  DZERO UW      U
   .
```

```
   .
   .

                     Process active channels
                     -----------------------

Channel  Window    Status  Device/file accessed
-------  ------    ------  --------------------
   0010  00000000          ROCK$DJA233:
   0020  8082C400          ROCK$DJA233:(1008,48490,0)
   0030  807F2260          LOVE$DUA200:(209,1,0)[V5COMMON.SYSLIB]SMGSHR.EXE;1 (section file)
   0040  00000000          VTA71:
   0050  00000000          VTA71:
   0060  807EFFE0          LOVE$DUA200:(195,1,0)[V5COMMON.SYSLIB]LIBRTL.EXE;1 (section file)
   0070  807EECC0          LOVE$DUA200:(199,1,0)[V5COMMON.SYSLIB]MTHRTL.EXE;1 (section file)
   0080  80838E80          LOVE$DUA200:(196,1,0)[V5COMMON.SYSLIB]LIBRTL2.EXE;1
   0090  807E4880          LOVE$DUA200:(210,1,0)[V5COMMON.SYSLIB]SORTSHR.EXE;1
   00A0  80818720          LOVE$DUA200:(191,1,0)[V5COMMON.SYSLIB]FDLSHR.EXE;1
   00B0  8083CFC0          LOVE$DUA200:(169,1,0)[V5COMMON.SYSLIB]CONVSHR.EXE;1
   00C0  8083DEC0          ROCK$DJA233:(1026,16,0)
```

The SHOW PROCESS/ALL command displays information taken from the software PCB of process GLOBE, and then proceeds to display the process header, the registers of the process, the process section table, the P0 page table, the P1 page table, and information about the I/O channels owned by the process. You can also obtain these displays by using the /PCB, /PHD, /REGISTERS, /PROCESS_SECTION_TABLE, /P0, /P1, and /CHANNEL qualifiers, respectively.

3.
```
SDA>  SHOW PROCESS/LOCKS/INDEX=0A


Lock data:

Lock id:  09960A0F   PID:    0001000A   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  00000000   Granted at    PW            SYSTEM
Sublocks:      100
LKB:      8082B0E0

Resource:      003C0248 24534D52    RMS$H.<.  Status:  ASYNC
 Length   26   444B4C4F 46020000    ...FOLKD
 Kernel mode   00202020 20202024    $      .
 System        00000000 00000000    ........
Local copy
```

The SHOW PROCESS/LOCKS/INDEX=0A command displays information about the locks held by process JOB_CONTROL, whose PCB is at index 0A, into the system's PCB list. This command implicitly makes JOB_CONTROL the SDA current process for subsequent commands that display process context information. It has no effect on SDA CPU context because JOB_ CONTROL is not current on any processor in the multiprocessing system.

4.
```
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> SHOW PROCESS/RMS
   .
   .
   .
Process index: 0032   Name: BEASSEM_MTHRTL_   Extended PID: 27200132
------------------------------------------------------------------------
IFAB Address: 7FF9C808  IFI: 0002       Organization: Sequential
------------

PRIM_DEV:       1C4D4108        DIR,FOD,SHR,AVL,ELG,IDV,ODV,RND
BKPBITS:        00080020        ACCESSED,NORECLK
BLN:            3A      58.             BID:            0B      11.
EFN:            00                      MODE:           03
IOS:            00000001                ASBADDR:        00000000
IOS2:           0000                    WAIT_Q_FLINK:   00000000
IOS4:           00000000                ARGLST:         7FF21418
ATJNLBUF:       00000000                WAIT_Q_BLINK:   00000000
FSBPTR:         00000000                AGENT_MODE:     03
SHR:            02      SHRGET
IRAB_LNK:       7FF9C958                CHNL:           00C0
FAC:            02      GET
ORGCASE:        00              Sequential
LAST_FAB:       00081FD0                NWA_PTR:        00000000
IFI:            0002                    ECHO_ISI:       0000
FWA_PTR:        7FF9CC00
BDB_FLNK:       7FF9CBB0                DEVBUFSIZ:      00000200     512.
BDB_BLNK:       7FF9CB60                RTDEQ:          0000           0.
RFMORG:         02              VAR
RAT:            02              CR
LRL:            004C          76.       HBK_DISK:       000C0000
FFB:            0084         132.       EBK_DISK:       000C0000
FSZ:            00             0.       BKS:            00             0.
DEQ:            0000           0.       MRS:            0000           0.
HBK:            0000000C      12.       GBC:            0000           0.
EBK:            0000000C
LAST_GOOD_EBK:  00000000       0.       LAST_GOOD_FFB:  0000           0.
RNS_LEN:        00000000                LOCK_BDB:       00000000

   .
   .
   .
```

The SHOW PROCESS/RMS command displays RMS data structures for the current SDA process.

5. SDA> SHOW PROCESS/IMAGES

```
                        Process activated images
                        ------------------------

ICB       Start     End       Type            Image Name  Major ID,Minor ID
--------  --------  --------  --------------  ----------------------------
7FF83878  00000200  00000DFF  MAIN            SHOW_PROC_IMAGES  0,0
7FF84100  0003AC00  0003FBFF  GLOBAL PRT SHR  DECW$TRANSPORT_COMMON  12,12
7FF84400  00036200  0003ABFF  GLOBAL          CONVSHR  1,0
7FF84470  0002E400  000361FF  GLOBAL          FDLSHR  1,0
7FF84560  00021A00  0002E3FF  GLOBAL          SORTSHR  2,28
7FF845D0  00000E00  000089FF  GLOBAL          LIBRTL2  1,12
7FF835F8  00008A00  000219FF  GLOBAL     SHR  LIBRTL  1,14
7FF84800  00060C00  000767FF  MERGED     SHR  ADARTL  0,0
7FF84720  00076800  000A03FF  GLOBAL     SHR  MTHRTL  129,32781

Total images = 9               Pages allocated = 1017
```

The SHOW PROCESS/IMAGES command 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.

6.
```
SDA> SHOW PROCESS/TRANSACTIONS=(DISPLAY=THREADS,
   TID=FAC21DE2-BA88-0092-8FA6-B24B)
```

The SHOW PROCESS command displays the transaction thread information for the transaction whose identifier is FAC21DE2-BA88-0092-8FA6-B24B.

SHOW RESOURCE

Displays information about all resources in the system or about a resource associated with a specific lock.

Format

SHOW RESOURCE {/ALL | /LOCKID=lock-id | /NAME=resource-name}

Parameters

None.

Qualifiers

/ALL
Displays information from all resource blocks (RSBs) in the system. This is the default behavior of the SHOW RESOURCE command.

/LOCKID=lock-id
Displays information about the resource associated with the lock with the specified **lock-id**.

/NAME=resource-name
Displays information about the resource whose resource name begins with the specified **resource-name**. For case-sensitive names, enclose **resource-name** in quotation marks.

Description

The SHOW RESOURCE command displays the information listed in Table SDA–21 for each resource in the system or for the specific resource associated with the specified **lock-id**.

Table SDA–21 Resource Information in the SHOW RESOURCE Display

Field	Contents
Address of RSB	Address of the resource block (RSB) that describes this resource.
Parent RSB	Address of the RSB that is the parent of this RSB. This field is 00000000 if the RSB itself is a parent block.
Sub-RSB count	Number of RSBs of which this RSB is the parent. This field is 0 if the RSB has no sub-RSBs.

(continued on next page)

Table SDA–21 (Cont.) Resource Information in the SHOW RESOURCE Display

Field	Contents
Group grant mode	Indication of the most restrictive mode in which a lock on this resource has been granted. This field can contain the following values (shown in order from the least restrictive mode to the most restrictive): • NL Null mode • CR Concurrent-read mode • CW Concurrent-write mode • PR Protected-read mode • PW Protected-write mode • EX Exclusive mode For information about conflicting and incompatible lock modes, see the *OpenVMS System Services Reference Manual*.
Conversion grant mode	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.
BLKAST count	Number of locks on this resource that have requested a blocking AST.
Value block	Hexadecimal dump of the 16-byte block value block associated with this resource.
Sequence #	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.
CSID	Cluster system identification number (CSID) of the node that owns the resource.

(continued on next page)

Table SDA–21 (Cont.) Resource Information in the SHOW RESOURCE Display

Field	Contents
Resource	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.
Length	Length in bytes of the resource name.
—	Processor mode of the name space in which this RSB resides.
—	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.
Granted queue	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.
Conversion queue	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.
Waiting queue	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.

Examples

1. `SDA> SHOW RESOURCE`

```
Resource database
-----------------
Address of RSB:  807F6120  Group grant mode:       NL
Parent RSB:      806EA180  Conversion grant mode:  NL
Sub-RSB count:           0  BLKAST count:           0
Value block:   806CE510 00000000 00000002 00000002   Seq. #:  00000008
Resource:      09ED7324 42313146    F11B$sí.
 Length   10   00000000 00000200    ........         CSID:  00020041
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........

Granted queue (Lock ID / Gr mode):
     006801AE  NL

Conversion queue (Lock ID / Gr/Rq mode):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode):
     *** EMPTY QUEUE ***


Address of RSB:  807EB9E0  Group grant mode:       PW
Parent RSB:      00000000  Conversion grant mode:  EX
Sub-RSB count:           0  BLKAST count:           1
Value block:   00000000 00000003 00000000 0000FFF2   Seq. #:  0000027F Not valid
Resource:      32245F24 44414853    SHAD$_$2
 Length   16   3A31534A 44243435    54$DJS1:         CSID:  0002001A
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........
   .
   .
   .
```

The SHOW RESOURCE command displays information taken from the RSBs of all resources in the system. For instance, the RSB at $807EB9E0_{16}$ is a parent block with no sub-RSBs. The most restrictive lock granted on this resource is in protected-write (PW) mode. There is a lock on the conversion queue waiting to be converted from PW mode to exclusive (EX) mode.

2.
```
SDA>  SHOW PROCESS/LOCKS

Process index: 001C   Name: STARTQ   Extended PID: 4800011C
------------------------------------------------------------
Lock data:

Lock id:  0117054F   PID:    0001001C   Flags:   VALBLK  SYNCSTS SYSTEM
Par. id:  00000000   Granted at    PW            NOQUOTA
Sublocks:        0
LKB:      808091A0
Resource:      45527624 42313146    F11B$vRE  Status:  NOQUOTA
 Length   18   20205241 4D323053    S02MAR
 Kernel mode   00000000 00002020      ......
 System        00000000 00000000    ........
Process copy of lock 008209CF on system 0002001
   .
   .
   .
SDA>  SHOW RESOURCE/LOCKID=117054F
```

```
Resource database
-----------------
Address of RSB:  806BB050  Group grant mode:       NL
Parent RSB:      00000000  Conversion grant mode:  NL
Sub-RSB count:          4  BLKAST count:            0
Value block:   00960102 0000330B 000735AA 5A020005   Seq. #:  00006D9F
Resource:      45527624 42313146    F11B$vRE
 Length   18   20205241 4D323053    S02MAR           CSID:  0002001A
 Kernel mode   00000000 00002020      ......
 System        00000000 00000000    ........

Granted queue (Lock ID / Gr mode):
     0117054F  PW       00060545  CR

Conversion queue (Lock ID / Gr/Rq mode):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode):
     *** EMPTY QUEUE ***
```

The SHOW PROCESS/LOCKS command lists all locks associated with the SDA current process, STARTQ. Its display is identical to that of the SHOW LOCK command, illustrated in Table SDA–15. The SHOW RESOURCE/LOCKID=117054F command determines that this particular lock is on the granted queue in protected-write mode for the resource at $806BB050_{16}$.

3.
```
SDA>  SHOW RESOURCE/NAME=RMS$

Resource database
-----------------

Address of RSB:  80EFBE40  GGMODE:       EX  Status: DIRENTR VALID
Parent RSB:      00000000  CGMODE:       EX
Sub-RSB count:          2  FGMODE:       EX
Lock Count:             1  CSID:   00000000
BLKAST count:           1  RQSEQNM:    0000

Resource:      00030014 24534D52   RMS$....  Valblk: 00000000 00000000
 Length   26   4D565841 56020000   ...VAXVM          00000000 00000000
 Exec. mode    00202035 35305653   SV055  .
 System        00000000 00000000   ........  Seqnum: 00000000

Granted queue (Lock ID / Gr mode / Range):
 6400004C  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 example of the SHOW RESOURCE/NAME command displays information about the resource whose name begins with RMS$.

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 SDA–14.

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=(IFB,CCB,WCB)
SDA>  SHOW RMS

RMS Display Options:  IFB,CCB,WCB
Display RMS structures for all IFI values.
```

   The SET RMS command establishes the IFB, CCB, and WCB as the structures to be displayed when you issue the SHOW PROCESS/RMS command. The SHOW RMS command verifies this selection of RMS options.

SHOW RSPID

Displays information about response IDs (RSPIDs) of all 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**.[14]

Description

Whenever a local system application (SYSAP) requires a response from a remote SYSAP, the local system assigns a unique number, called an RSPID, to the response. 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 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)

[14] 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.

Examples

1. SDA> SHOW RSPID

```
VAXcluster data structures
--------------------------

    --- Summary of Response Descriptor Table(RDT) 8037A4A8 ---

RSPID      CDRP Address     CDT Address     Local Process Name      Remote Node
-----      ------------     -----------     ------------------      -----------

04C30000     803917B0        8037AB50       VMS$DISK_CL_DRVR        SOWHAT
06260001     80804FA0        8037AF10       VMS$VAXcluster          WALKIN
0C390002     807E0460        8037AD30       VMS$VAXcluster          OLEO
   .
   .
   .
```

The SHOW RSPID command shows the response IDs that are currently open for all local connections in the VAXcluster system.

2. SDA> SHOW RSPID/CONNECTION=G37B7D0

```
VAXcluster data structures
--------------------------

    --- Summary of Response Descriptor Table(RDT) 8037A4A8 ---

RSPID      CDRP Address     CDT Address     Local Process Name      Remote Node
-----      ------------     -----------     ------------------      -----------

08B8001C     807F0300        8037B7D0       VMS$VAXcluster          METEOR
0915001D     807F08A0        8037B7D0       VMS$VAXcluster          METEOR
```

The SHOW RSPID/CONNECTION=G37B7D0 command displays only those RSPIDs in use that are associated with the SCS connection whose CDT is at address $8037B7D0_{16}$.

SHOW SPINLOCKS

Displays information taken from the data structures that provide system synchronization in a multiprocessing environment.

The default qualifiers are /STATIC and /DYNAMIC.

Format

SHOW SPINLOCKS [/OWNED][/BRIEF | /FULL][/DYNAMIC | /STATIC]
[name | /ADDRESS=expression | /INDEX=expression]

Parameter

name
Name of the spin lock, fork lock, or device lock structure to be displayed. You can obtain the names of the static system spin locks and fork locks from Table SDA–22. Device lock names are of the form *[node$]lock*, where *node* optionally indicates the VAXcluster node name (allocation class) and *lock* indicates the device and controller identification (for example, HAETAR$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, spin lock name or device name, IPL or device IPL, rank, index, ownership depth, number of waiting CPUs, CPU ID of the owner CPU, and interlock status (depth of ownership).

/DYNAMIC
Displays information for all device locks in the system.

/FULL
Displays full descriptive and diagnostic information for each displayed spin lock, fork lock, or device lock.

/INDEX=expression
Displays the system spin lock whose index is specified in **expression**. You cannot use the /INDEX qualifier to display a device lock.

/OWNED
Displays information for all spin locks, fork locks, and device locks owned by the SDA current CPU. If a processor does not own any spin locks, SDA displays the following message:

```
No spinlocks currently owned by CPU xx
```

The *xx* represents the CPU ID of the processor.

/STATIC
Displays information for all system spin locks and fork locks.

Description

The SHOW SPINLOCKS command displays status and diagnostic information about the multiprocessing synchronization structures known as spin locks.

A **static spin lock** is a spin lock whose data structure is permanently assembled into the system. Static spin locks are accessed as indexes into a vector of longword addresses called the **spin lock vector**, the address of which is contained in SMP$AR_SPNLKVEC. System spin locks and fork locks are static spin locks. Table SDA–22 lists the static spin locks.

A **dynamic spin lock** is a spin lock that is created based on the configuration of a particular system. One such dynamic spin lock is the device lock SYSGEN creates when configuring a particular device. This device lock synchronizes access to the device's registers and certain UCB fields. The operating system creates a dynamic spin lock by allocating space from nonpaged pool, rather than assembling the lock into the system as it does in creating a static spin lock.

See the *OpenVMS VAX Device Support Manual*[15] for a full discussion of the role of spin locks in maintaining synchronization of kernel mode activities in a multiprocessing environment.

Table SDA–22 Static Spin Locks

Name	Description
QUEUEAST	Fork lock for queuing ASTs at IPL 6
FILSYS	Lock on file system structures
IOLOCK8	Fork lock for executing a driver fork process at IPL 8
PR_LK8	Primary CPU's private lock for IPL 8
TIMER	Lock for adding and deleting timer queue entries and searching the timer queue
JIB	Lock for manipulating job nonpaged pool quotas as reflected by the fields JIB$L_BYTCNT and JIB$L_BYTLM in the job information block (JIB)
MMG	Lock on memory management, PFN database, swapper, modified page writer, and creation of per-CPU database structures
SCHED	Lock on process control blocks (PCBs), scheduler database, and mutex acquisition and release structures
IOLOCK9	Fork lock for executing a driver fork process at IPL 9
PR_LK9	Primary CPU's private lock for IPL 9
IOLOCK10	Fork lock for executing a driver fork process at IPL 10
PR_LK10	Primary CPU's private lock for IPL 10
IOLOCK11	Fork lock for executing a driver fork process at IPL 11
PR_LK11	Primary CPU's private lock for IPL 11

(continued on next page)

[15] This manual has been archived but is available on the OpenVMS Documentation CD-ROM.

Table SDA–22 (Cont.) Static Spin Locks

Name	Description
MAILBOX	Lock for sending messages to mailboxes
POOL	Lock on nonpaged pool database
PERFMON	Lock for I/O performance monitoring
INVALIDATE	Lock for system space translation buffer (TB) invalidation
VIRTCONS	Lock for ownership of the virtual console
HWCLK	Lock on hardware clock database, including the quadword containing the due time of the first timer queue entry (EXE$GQ_1ST_TIME) and the quadword containing the system time (EXE$GQ_SYSTIME)
MEGA	Lock for serializing access to fork-wait queue
MCHECK	Lock for synchronizing certain machine error handling
EMB	Lock for allocating and releasing error logging buffers

Note

The MCHECK and EMB spin locks, formerly separate spin locks in previous releases of OpenVMS, have been merged. When you analyze a crash, you might see one or both names when you display static spin locks.

For each spin lock, fork lock, or device lock in the system, SHOW SPINLOCKS provides the following information:

- Name of the spin lock (or device name for the device lock)
- Address of the spin lock data structure (SPL)
- The owner CPU's CPU ID
- IPL at which allocation of the lock is synchronized on a local processor
- Number of nested acquisitions of the spin lock by the processor owning the spin lock ("Ownership Depth")
- Rank of the spin lock
- Number of processors waiting to obtain the spin lock
- Spin lock index (for static spin locks only)
- Timeout interval for spin lock acquisition (in terms of 10 milliseconds)

SHOW SPINLOCKS/BRIEF produces a condensed display of this same information.

If the system under analysis was executing with full-checking multiprocessing enabled (according to the setting of the MULTIPROCESSING system parameter), SHOW SPINLOCKS/FULL adds to the spin lock display the last eight 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.

Examples

1. SDA> SHOW SPINLOCKS

```
System static spinlock structures
---------------------------------
EMB                               Address : 801B9EF8
Owner CPU ID     : None           IPL     : 1F
Ownership Depth  : 0000           Rank    : 00
CPUs Waiting     : 0000           Index   : 20
Timeout interval 002DC60

MCHECK                            Address : 801B9F48
Owner CPU ID     : None           IPL     : 1F
Ownership Depth  : 0000           Rank    : 01
CPUs Waiting     : 0000           Index   : 21
Timeout interval 002DC60
   .
   .
   .
IOLOCK8                           Address : 801BA538
Owner CPU ID     : 02             IPL     : 08
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000           Index   : 34
Timeout interval 002DC60
   .
   .
   .
System dynamic spinlock structures
----------------------------------
HAETAR$MBA                        Address : 801BA178
Owner CPU ID     : None           IPL     : 0B
Ownership Depth  : 0000           Rank    : 08
CPUs Waiting     : 0000           Index   : 28
Timeout interval 002DC60

HAETAR$NLA                        Address : 801BA178
Owner CPU ID     : None           IPL     : 08
Ownership Depth  : 0000           Rank    : 08
CPUs Waiting     : 0000           Index   : 28
Timeout interval 002DC60

HAETAR$PAA                        Address : 8063A620
Owner CPU ID     : 02             DIPL    : 14
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000
Timeout interval 002DC60
   .
   .
   .
```

This excerpt illustrates the default output of the SHOW SPINLOCKS command. Note that the CPU whose CPU ID is 2 owns the fork lock IOLOCK8. CPU 2 must have an IPL of at least 8, which is the acquisition IPL of the fork lock. CPU 2 has no nested ownership of the fork lock. The rank of IOLOCK8 is 14_{16}, indicating that CPU 2 could not own any locks with a logical rank of 15_{16} or higher when it acquired IOLOCK8.

Similarly, while owning IOLOCK8, CPU 2 cannot obtain any additional spin locks with a logical rank of 14_{16} or lower.

No CPUs are waiting for the fork lock; its index is 34_{16}.

2. SDA> SHOW SPINLOCKS/BRIEF

```
Address  Spinlock Name    IPL Rank Index Depth #Waiting  Owner CPU Interlock
-------------------------------------------------------------------------------
801B9EF8 EMB              1F   00    20   00     0000      None       Free
801B9EF8 MCHECK           1F   00    20   00     0000      None       Free
801B9F98 MEGA             1F   02    22   00     0000      None       Free
801B9FE8 HWCLK            16   03    23   00     0000      None       Free
801BA038 VIRTCONS         14   04    24   00     0000      None       Free
801BA088 INVALIDATE       13   05    25   00     0000      None       Free
801BA0D8 PERFMON          0F   06    26   00     0000      None       Free
801BA128 POOL             0B   07    27   00     0000      None       Free
801BA178 MAILBOX          0B   08    28   00     0000      None       Free
801BA1C8 PR_LK11          0B   09    29   00     0000      None       Free
801BA218 IOLOCK11         0B   0A    2A   00     0000      None       Free
801BA268 PR_LK10          0A   0B    2B   00     0000      None       Free
801BA2B8 IOLOCK10         0A   0C    2C   00     0000      None       Free
801BA308 PR_LK9           09   0D    2D   00     0000      None       Free
801BA358 IOLOCK9          09   0E    2E   00     0000      None       Free
801BA3A8 SCHED            08   0F    2F   00     0000      None       Free
801BA3F8 MMG              08   10    30   00     0000      None       Free
801BA448 JIB              08   11    31   00     0000      None       Free
801BA498 TIMER            08   12    32   00     0000      None       Free
801BA4E8 PR_LK8           08   13    33   00     0000      None       Free
801BA538 IOLOCK8          08   14    34   01     0000        02         00
801BA588 FILSYS           08   15    35   00     0000      None       Free
801BA5D8 QUEUEAST         06   16    36   00     0000      None       Free
8016A628 ASTDEL           02   17    37   00     0000      None       Free

Address  Device Name     DIPL Rank Index Depth #Waiting  Owner CPU Interlock
-------------------------------------------------------------------------------
801BA178 HAETAR$MBA       0B   08    28   00     0000      None       Free
801BA178 HAETAR$NLA       08   08    28   00     0000      None       Free
8063A620 HAETAR$PAA       14   14         01     0000        02         00
8063C5C0 HAETAR$XEA       15   FF         00     0000      None       Free
8063C4A0 HAETAR$XGA       15   FF         00     0000      None       Free
8063C380 HAETAR$PEA       14   FF         00     0000      None       Free
8063AC40 HAETAR$TXA       15   FF         00     0000      None       Free
8063A520 HAETAR$LCA       15   FF         00     0000      None       Free
801BA538 HAETAR$CNA       08   14    34   01     0000        02         00
   .
   .
   .
```

This excerpt illustrates the condensed form of the display produced in the first example.

3. SDA> SHOW SPINLOCKS/OWNED

```
System static spinlock structures
---------------------------------
IOLOCK8                           Address : 801BA538
Owner CPU ID     : 02             IPL     : 08
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000           Index   : 34
Timeout interval 002DC60
   .
   .
   .
System dynamic spinlock structures
----------------------------------
HAETAR$PAA                        Address : 8063A620
Owner CPU ID     : 02             DIPL    : 14
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000
Timeout interval 002DC60

HAETAR$CNA                        Address : 801BA538
Owner CPU ID     : 02             IPL     : 08
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000           Index   : 34
Timeout interval 002DC60

HAETAR$NET                        Address : 801BA538
Owner CPU ID     : 02             IPL     : 08
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000           Index   : 34
Timeout interval 002DC60

HAETAR$NDA                        Address : 801BA538
Owner CPU ID     : 02             IPL     : 08
Ownership Depth  : 0001           Rank    : 14
CPUs Waiting     : 0000           Index   : 34
Timeout interval 002DC60
   .
   .
   .
```

The SHOW SPINLOCKS/OWNED command shows all owned spin locks in the system.

4. SDA> SHOW SPINLOCKS/FULL

```
System static spinlock structures
---------------------------------
EMB                                 Address : 801B9EF8
Owner CPU ID     : None             IPL     : 1F
Ownership Depth  : 0000             Rank    : 00
CPUs Waiting     : 0000             Index   : 20
Timeout interval 002DC60

Spinlock EMB was last acquired or released from:
(Most recently)             80195146 ERL$WAKE+00089
       .                    801950EF ERL$WAKE+00032
       .                    80195146 ERL$WAKE+00089
       .                    801950EF ERL$WAKE+00032
       .                    80195146 ERL$WAKE+00089
       .                    801950EF ERL$WAKE+00032
       .                    80195146 ERL$WAKE+00089
(Least recently)            801950EF ERL$WAKE+00032
   .
   .
   .
Last release of multiple acquisitions occurred at:
                            801194F9 EXE$INSIOQ+00044
   .
   .
   .
IOLOCK8                             Address : 801BA538
Owner CPU ID     : 02               IPL     : 08
Ownership Depth  : 0001             Rank    : 14
CPUs Waiting     : 0000             Index   : 34
Timeout interval 002DC60

Spinlock IOLOCK8 was last acquired or released from:
(Most recently)             801BBE08 EXE$FORKDSPTH+0007E
       .                    80198EBF EXE$QIOACPPKT+00052
       .                    80198E7E EXE$QIOACPPKT+00011
       .                    80199BB2 IOC$CHECK_HWM+0032D
       .                    80182DE5 LCK$QUEUED_EXIT+0001D
       .                    80182884 LCK$AR_COMPAT_TBL+0007C
       .                    8018357E EXE$DEQ+00189
(Least recently)            80183428 EXE$DEQ+00033
   .
   .
   .
```

The SHOW SPINLOCKS/FULL command displays a list of the last eight PCs that have accessed the spin lock. For instance, the fork dispatcher contains the code that most recently acquired the fork lock.

SHOW STACK

Displays the location and contents of the four process stacks of the SDA current process and the interrupt stack of the SDA current CPU.

Format

SHOW STACK [range | /qualifier[,...]]

Parameters

range
Range of memory locations you want to display in stack format. You can express a **range** using the following format:

m:n Range of virtual addresses from *m* to *n*

m;n Range of virtual addresses starting at *m* and continuing for *n* bytes

Qualifiers

/ALL
Displays the locations and contents of the four process stacks for the SDA current process and the interrupt stack for the SDA current CPU.

/EXECUTIVE
Shows the executive stack for the SDA current process.

/INTERRUPT
Shows the interrupt stack for the SDA current CPU.

/KERNEL
Shows the kernel stack for the SDA current process.

/SUPERVISOR
Shows the supervisor stack for the SDA current process.

/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 any other process made the SDA current process, the SHOW STACK command by default shows its current operating stack.

The various qualifiers to the command can display any of the four per-process stacks for the SDA current process, as well as the interrupt stack for the SDA current CPU.

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 Section 4.

SDA provides the following information in each stack display.

Section	Contents
Identity of stack	SDA indicates whether the stack is a process stack (user, supervisor, executive, or kernel) or the processor interrupt stack. If the interrupt stack is being displayed, SDA displays the CPU ID of the processor that owns it. Similarly, if the SDA current process is currently scheduled on a processor in the system, SHOW STACK also specifies the CPU ID of the processor on which the process is scheduled.
Stack pointer	The stack pointer identifies the top of the stack. The display indicates the stack pointer by the symbol SP =>.
Stack address	SDA lists all the virtual addresses that the operating system has allocated to the stack. The stack addresses are listed in a column that increases in increments of 4 bytes (one longword).
Stack contents	SDA lists the contents of the stack in a column to the right of the stack addresses.
Symbols	SDA attempts to display the contents of a location symbolically, using a symbol and an offset. If the address is not within FFF_{16} of the value of any existing symbol, this column is left blank.

If a stack is empty, the display shows the following:

```
SP =>  (STACK IS EMPTY)
```

Example

```
SDA> SHOW STACK
Process stacks (on CPU 00)
--------------------------

Current operating stack (USER):

               7FF73278  200C0000
               7FF7327C  00001518      SGN$C_MAXPGFL+518
               7FF73280  7FF732F0
               7FF73284  000187A7      RMS$_ECHO+72E

        SP =>  7FF73288  0000060A      BUG$_NOHDJMT+002
               7FF7328C  00000000
               7FF73290  00000003
               7FF73294  7FF73800
               7FF73298  7FF73800
```

The SHOW STACK command displays a user stack that was the current operating stack for a process scheduled on CPU 00. The data shown above the stack pointer might not be valid. The symbol to the right of the columns, BUG$_ NOHDJMT+002, is the result of the SDA attempt to interpret the contents of the longword at the top of the stack as a symbol meaningful to the user. In this case, the value on the stack and the value of BUG$_NOHDJMT are unrelated.

SHOW SUMMARY

Displays a list of all active processes and the values of the parameters used in swapping and scheduling those processes.

Format

SHOW SUMMARY [/IMAGE]

Parameters

None.

Qualifier

/IMAGE
Causes SDA to display, if possible, the name of the image being executed within each process.

Description

The SHOW SUMMARY command displays the information in Table SDA–23 for each active process in the system.

Table SDA–23 Process Information in the SHOW SUMMARY Display

Column	Contents
Extended PID	32-bit number that uniquely identifies the process
Indx	Index of this process into the PCB array
Process name	Name assigned to the process
Username	Name of the user who created the process

(continued on next page)

Table SDA–23 (Cont.) Process Information in the SHOW SUMMARY Display

Column	Contents
State	Current state of the process, one of the following 14 states: • COM Computable and resident in memory • COMO Computable but outswapped • CUR Currently executing[1] • CEF Waiting for a common event flag • LEF Waiting for a local event flag • LEFO Outswapped and waiting for a local event flag • HIB Hibernating • HIBO Hibernating and outswapped • SUSP Suspended • SUSPO Suspended and outswapped • PFW Waiting for a page that is not in memory (page-fault wait) • FPG Waiting to add a page to its working set (free-page wait) • COLPG 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 • MWAIT Waiting for a system resource (miscellaneous wait)
Pri	Current scheduling priority of the process

[1]For a process in the CUR state executing in a multiprocessing environment, SDA indicates the CPU ID of the processor on which the process is current. This information, however, might not be accurate in SHOW SUMMARY displays produced in the analysis of a running system.

(continued on next page)

Table SDA–23 (Cont.) Process Information in the SHOW SUMMARY Display

Column	Contents
PCB	Address of the process control block
PHD	Address of the process header
Wkset	Number (in decimal) of pages currently in the working set of the process

Example

```
SDA> SHOW SUMMARY/IMAGE

Current process summary
-----------------------
Extended  Indx Process name    Username    State   Pri   PCB      PHD    Wkset
-- PID -- ---- -------------- ----------- ------- --- -------- -------- -----
 33C00101 0001 SWAPPER                     HIB     16 8000C3C0 8000C200     0
 33C00205 0005 _RTA5:          SIVAD       LEF      4 80482FE0 82120E00   293
 33C00106 0006 ERRFMT          SYSTEM      HIB      8 80432950 80DB4600   126
          $254$DUA200:[SYS6.SYSCOMMON.][SYSEXE]ERRFMT.EXE;1
 33C00107 0007 CACHE_SERVER    SYSTEM      HIB     16 80432AC0 81121E00   120
          $254$DUA200:[SYS6.SYSCOMMON.][SYSEXE]FILESERV.EXE;400
 33C00108 0008 CLUSTER_SERVER  SYSTEM      HIB     10 804331F0 81246600   313
          $254$DUA200:[SYS6.SYSCOMMON.][SYSEXE]CSP.EXE;300
   .
   .
   .
 33C0010D 000D NETACP          DECNET      CUR  00 10 8044C6D0 816D8600  1500
          $254$DUA200:[SYS6.SYSCOMMON.]<SYSEXE>NETACP.EXE;3
 33C0010E 000E EVL             DECNET      HIB      4 8044CD60 817FCE00    68
          $254$DUA200:[SYS6.SYSCOMMON.]<SYSEXE>EVL.EXE
   .
   .
   .
```

The SHOW SUMMARY/IMAGE command describes all active processes in the system at the time of the system failure. Note that the process NETACP is in the CUR state on CPU 00 of a multiprocessor system 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] symbol-name

Parameter

symbol-name
Name of the symbol to be displayed. You must provide a **symbol-name**.

Qualifier

/ALL
Displays information about all symbols whose names begin with the characters specified in **symbol-name**.

Description

The SHOW SYMBOL/ALL command is useful for determining the values of symbols that belong to a symbol set, as illustrated in the examples.

Examples

1.
```
SDA> SHOW SYMBOL G
G = 80000000 :  8FBC0FFC
```

The SHOW SYMBOL command evaluates the symbol G as 80000000_{16} and displays the contents of address 80000000_{16} as $8FBC0FFC_{16}$.

2.
```
SDA>  SHOW SYMBOL/ALL BUG


Symbols sorted by name
----------------------
BUG$BUILD_HEADE 80002038 => 24A89F16       BUG$_CONSOLRX50 00000640 => 10A2020E
BUG$DUMP_REGIST 80002040 => 24A89F16       BUG$_CONTRACT   000000C0
BUG$FATAL       80002048 => 24A89F16       BUG$_CPUBUSYWAI 00000780 => 6501FB30
BUG$L_BUGCHK_FL 80004108 => 00000001       BUG$_CPUCEASED  000005E8 => 5EDD0000
BUG$L_FATAL_SPS 8000410C => 7FFE7C6C       BUG$_CPUEXIT    000006B8 => 218FD007
BUG$READ_ERR_RE 80002050 => 24A89F16       BUG$_CPUSANITY  00000778 => 8A031164
BUG$REBOOT      80002058 => 6E9E9F17       BUG$_CTERM      00000678 => 00000004
BUG$TABLE       8000D09E => 00280001       BUG$_CWSERR     00000698 => 004C414E
   .
   .
   .
```

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 TRANSACTIONS

Displays information about all transactions on the node or about a specified transaction.

Format

SHOW TRANSACTIONS [/qualifier[,...]]

Qualifiers

/DISPLAY=(item [,...])
Specifies the type of information to be displayed. The argument to /DISPLAY can be either a single item or a list. The following items can be specified.

Item	Description
ALL	All transaction control structures for the specified transaction. This is the default behavior.
BRANCHES	Control structures for branches of the specified transaction.
PARTICIPANTS	Control structures for resource managers participating in the specified transaction.
THREADS	Control structures for threads of the specified transaction.
TRANSACTIONS	Transaction control structures for the specified transaction.

/SUMMARY
Displays statistics for transactions on the node. The /SUMMARY qualifier cannot be used with the /TID or /DISPLAY qualifier.

/TID=tid
Specifies the transaction for which information is to be displayed. If you omit the /TID qualifier, the SHOW TRANSACTIONS command displays information about all transactions on the node.

Examples

1. `SDA> SHOW TRANSACTIONS/TID=FAC21DE2-BA88-0092-8FA6-00000000B24B`

   The SHOW TRANSACTIONS command displays all the transaction control structure information for the transaction identified by the transaction identifier.

2. `SDA> SHOW TRANSACTIONS/DISPLAY=(PARTICIPANTS, BRANCHES)`

   The SHOW TRANSACTIONS command displays the transaction branch and resource manager information for all transactions on the node.

SPAWN

Creates a subprocess of the process currently running SDA, copying the context of the current process to the subprocess and, optionally, executing within the subprocess a specified command.

Format

SPAWN [/qualifier[,...]] [command]

Parameter

command
Name of the command that you want executed by the subprocess.

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$OUTPUT when the subprocess completes processing or aborts. The default behavior is that such a message is not sent to SYS$OUTPUT.

When you use this qualifier, you must also specify the /NOWAIT qualifier.

/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, you should 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. You should specify an output other than SYS$OUTPUT whenever you specify /NOWAIT to prevent output from the spawned subprocess from being displayed while you are specifying new commands. If you omit the /OUTPUT qualifier, output is written to the current SYS$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.

Example

```
SDA>  SPAWN
$  MAIL
   .
   .
   .
$  DIR
   .
   .
   .
$  LO
   Process SYSTEM_1 logged out at 5-MAR-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.

VALIDATE QUEUE

Validates the integrity of the specified queue by checking the pointers in the queue.

Format

VALIDATE QUEUE [address] [/qualifier[,...]]

Parameter

address
Address of an element in a queue.

If you specify a 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

/MAXIMUM_LINKS=nn
Specifies the number of entries in the queue that are to be validated.

/SELF_RELATIVE
Specifies that the selected queue is a self-relative queue.

Description

The VALIDATE QUEUE command uses the forward and 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 the VALIDATE QUEUE command is used 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 SCH$GQ_LEFWQ/MAXIMUM_LINKS=3
The queue is consistent through 3 elements
```

   This example validates three elements in the SCH$GQ_LEFWQ queue.

2.
```
SDA>  VALIDATE QUEUE/SELF_RELATIVE IOC$GL_IRPFL
Queue is complete, total of 159 elements in the queue
```

   This example validates the self-relative queue that is the IRP pool list. The validation is successful and determines that there are 159 IRPs in the list.

Index

A

Access violations, SDA–21, SDA–23
ACP (ancillary control process), SDA–104
Addition operator (+), SDA–16
Addresses, examining, SDA–53
/ADDRESS qualifier, SDA–90, SDA–103, SDA–142
/ALL qualifier, SDA–53, SDA–121, SDA–131, SDA–149, SDA–161, SDA–181
  SHOW PAGE_TABLE command, SDA–126
  SHOW STACK command, SDA–176
AMB symbol, SDA–17
ANALYZE/CRASH_DUMP/RELEASE command, SDA–5
ANALYZE/CRASH_DUMP command, SDA–9, SDA–35
ANALYZE/SYSTEM command, SDA–3, SDA–35
ANALYZE command, SDA–35
  /CRASH_DUMP qualifier, SDA–37
  /RELEASE qualifier, SDA–38
  /SYMBOL qualifier, SDA–39
  /SYSTEM qualifier, SDA–40
Analyzing a crash dump
  See Crash dumps
  See System failures
Analyzing a running system, SDA–11, SDA–40
  privileges required, SDA–11, SDA–35
AND operator (&), SDA–16
AP (argument pointer), SDA–17
AP symbol, SDA–17
AQBs (ACP queue blocks), SDA–105
Arithmetic operators, SDA–16
  shifting (@), SDA–17
ASBs (asynchronous save blocks), SDA–79
ASTLVL register, displaying, SDA–95
AST routines, global symbols, SDA–63
ATTACH command, SDA–45

B

Backup utility (BACKUP), copying system dump file, SDA–7
Bad page list, displaying, SDA–131
/BAD qualifier, SDA–131
BDBs (buffer descriptor blocks), SDA–79
BDBSUM (BDB summary page), SDA–79
Binary operators, SDA–16 to SDA–17
BLBs (buffer lock blocks), SDA–79
BLBSUM (BLB summary page), SDA–79
Bugchecks
  code, SDA–19
  fatal conditions, SDA–20 to SDA–24
  global symbols, SDA–63
  halt/restart, SDA–9
  handling routines, SDA–63
  identifying, SDA–25
  reasons for taking, SDA–99
/BUS qualifier, SDA–142

C

/CACHED qualifier, SDA–121
Call frames
  displaying in SDA, SDA–82
  following a chain, SDA–82
Cancel I/O routine, SDA–104
CCBs (channel control blocks), displaying in SDA, SDA–79
CDDBs (class driver data blocks), SDA–105
CDDB symbol, SDA–17
CDRPs (class driver request packets), SDA–90, SDA–167
CDTs (connection descriptor tables), SDA–167
  displaying contents, SDA–90
  displaying SDA information, SDA–90
/CHANNEL qualifier, SDA–142, SDA–154
CLUBs (cluster blocks), SDA–86
CLUDCBs (cluster quorum disk control blocks), SDA–86
CLUFCBs (cluster failover control blocks), SDA–86
Cluster management code, global symbols, SDA–63
CLUSTRLOA.STB file, SDA–63
CLUSTRLOA symbol, SDA–17
Condition-handling routines, global symbols, SDA–63
Condition values
  evaluating, SDA–51
  examining, SDA–53

/CONDITION_VALUE qualifier, SDA–51
Connection manager, displaying SDA information, SDA–85
/CONNECTION qualifier, SDA–167
Connections
  displaying SDA information about, SDA–142, SDA–167
Connections, displaying SDA information about, SDA–90
Context
  SDA CPU, SDA–14
  SDA process, SDA–12
Control blocks, formatting, SDA–58
Control region, SDA–18
  base register, SDA–18
  examining, SDA–54
  length register, SDA–18
  page table, displaying, SDA–150
Control region operator (H), SDA–16
COPY command, SDA–5, SDA–6, SDA–46
CPU context
  changing, SDA–94
    SDA current, SDA–71
    using the SET PROCESS command, SDA–77
    using the SHOW CPU command, SDA–94
    using the SHOW CRASH command, SDA–98
    using the SHOW PROCESS command, SDA–149
  displaying, SDA–94
CPU identification number, SDA–94
CPULOA.EXE file, global symbols, SDA–63
Crash dumps
  See System failures
  file headers, SDA–112
  incomplete, SDA–9
  privileges required, SDA–35
  requirements, SDA–8
  short, SDA–9
/CRASH_DUMP qualifier, SDA–9
CRBs (channel request blocks), SDA–104
CRB symbol, SDA–17
CREATE command, SDA–4
CSBs (cluster system blocks), SDA–85, SDA–90
/CSID qualifier, SDA–85
CSIDs (cluster system identification numbers), SDA–85, SDA–162
Current location symbol (.), SDA–17

D

Data structures
  formatting, SDA–58
  stepping through a linked list, SDA–67
DCLDEF.STB file, SDA–63
DCL interpreter, global symbols, SDA–63
DDBs (device data blocks), SDA–104
DDB symbol, SDA–17
DDTs (driver dispatch tables), SDA–104
DDT symbol, SDA–17
Decimal value of an expression, SDA–51
DECnet data structures, global symbols, SDA–63
DEFINE command, SDA–47
Device driver routines, address, SDA–104
Device drivers
  base address of driver prologue table (DPT), SDA–18
  locating, SDA–18
  locating a failing instruction, SDA–27
/DEVICE qualifier, SDA–142
Devices, displaying SDA information, SDA–103
Division operator (/), SDA–17
DPT base address, SDA–27
DPTs (driver prologue tables), SDA–104
DRIVER symbol
  See nnDRIVER symbol
DUMPBUG system parameter, SDA–4, SDA–32
Dump files
  analyzing, SDA–35
  copying the contents, SDA–46
DUMPSTYLE system parameter, SDA–6
DUMP subset, SDA–6

E

/ECHO qualifier, DEFINE command, SDA–48
ERRORLOG.EXE file, SDA–63
ERRORLOGBUFFERS system parameter, SDA–4
Error logging
  global symbols, SDA–63
  routines, SDA–63
ESP symbol, SDA–18
EVALUATE/PSL command, SDA–26
EVALUATE command, SDA–51
Event flag routines, global symbols, SDA–63
EVENT_FLAGS_AND_ASTS.EXE file, global symbols, SDA–63
EXAMINE/INSTRUCTION command, SDA–26
EXAMINE command, SDA–20, SDA–28, SDA–53
EXCEPTION.EXE file, global symbols, SDA–63
Exception-handling routines, global symbols, SDA–63
Exceptions
  fatal, SDA–20
  identifying causes of, SDA–25
Execute procedure (@) command, SDA–44
Executive images
  contents, SDA–63, SDA–110
  global symbols, SDA–62

/EXECUTIVE qualifier, SDA–62, SDA–176
Executive stack pointer, SDA–18
EXIT command, SDA–57
Expressions, SDA–15, SDA–19
Expressions, evaluating, SDA–51

F

FABs (file access blocks), SDA–79
Fatal exceptions, SDA–20
FATALEXCPT bugcheck, SDA–21
FCBs (file control blocks), SDA–79
Floating-point emulation code, base address, SDA–18
FORMAT command, SDA–29, SDA–58, SDA–67
FPEMUL symbol, SDA–18
FP symbol, SDA–18
Frame pointers, SDA–18
Free page list, displaying, SDA–131
/FREE qualifier, SDA–131, SDA–135
FWAs (file work areas), SDA–79

G

GBDs (global buffer descriptors), summary page, SDA–79
GBHs (global buffer headers), SDA–79
GBSBs (global buffer synchronization blocks), SDA–79
Global page tables, displaying, SDA–126
/GLOBAL qualifier, SDA–126
G operator, SDA–16
G symbol, SDA–18

H

/HEADER qualifier, SDA–135
HELP command, SDA–60
HELP command, recording output, SDA–74
Hexadecimal value of an expression, SDA–51
H operator, SDA–16
H symbol, SDA–18

I

I/O databases, displaying SDA information, SDA–103
ICCS register, displaying, SDA–95
IDBs (interrupt dispatch blocks), SDA–104
/ID qualifier, SDA–149
IDXs (index descriptors), SDA–79
IFABs (internal file access blocks), SDA–79
IFIs (internal file identifiers), SDA–79
/IF_STATE qualifier, SDA–48
Image activator
  global symbols, SDA–63
Image activator, global symbol, SDA–63
Image I/O structures, SDA–80
/IMAGE qualifier, SDA–178
/IMAGES qualifier, SDA–149
IMAGE_MANAGEMENT.EXE file, global symbols, SDA–63
IMGDEF.STB file, SDA–63
/INDEX qualifier, SDA–76, SDA–149
/INPUT qualifier, SDA–183
/INSTRUCTION qualifier, on EXAMINE command, SDA–53
Interlocked queues, validating, SDA–185
/INTERRUPT qualifier, SDA–176
Interrupt stack, displaying contents, SDA–176
INVEXCEPTN bugcheck, SDA–21
IO_ROUTINES.EXE file, global symbols, SDA–64
IPL$_ASTDEL value, PGFIPLHI bugcheck, SDA–23
IRABs (internal record access blocks), SDA–79
IRPs (I/O request packets), SDA–104
IRP symbol, SDA–18

J

JFBs (journaling file blocks), SDA–79
JIBs (job information blocks), SDA–152
JIB symbol, SDA–18

K

/KERNEL qualifier, SDA–176
Kernel stacks
  displaying contents, SDA–176
  pointer, SDA–18
/KEY qualifier, SDA–48
Keys (in records), defining for SDA, SDA–47
KSP symbol, SDA–18

L

Linker map, use in crash dump analysis, SDA–20
LKBs (lock blocks)
  definition, SDA–122
  displaying only cached, SDA–121
LMF$GROUP_TABLE.EXE file, global symbols, SDA–64
LNM symbol, SDA–18
Location in memory
  examining, SDA–53
  SDA default, SDA–53
  translating to MACRO instruction, SDA–53
/LOCKID qualifier, SDA–161
LOCKING.EXE file, SDA–64
Lock management routines, global symbols, SDA–64

Lock manager, displaying SDA information, SDA–121
Lock mode, SDA–162
Locks, displaying SDA information, SDA–161
/LOCKS qualifier, SDA–150
Logical operators, SDA–16
  AND (&), SDA–16
  NOT (#), SDA–16
  OR (|), SDA–16
  XOR (\ ), SDA–16
LOGICAL_NAMES.EXE file, global symbols, SDA–64

M

MA780 multiport memory, configuring a dump file for, SDA–5
Machine check code, base address, SDA–18
MACRO instruction, formatting memory with SDA, SDA–53
Mathematical operators, SDA–16
MCHK symbol, SDA–18
Mechanism arrays, SDA–21, SDA–25
Memory
  contents of a block
    formatting, SDA–58
  locations
    decoding, SDA–55
    examining, SDA–53, SDA–54
  regions, SDA–56
/MESSAGE qualifier, SDA–142
MESSAGE_ROUTINES.EXE file, global symbols, SDA–64
Modified page list, displaying, SDA–131
/MODIFIED qualifier, SDA–131
Modules, finding failing, SDA–27
MSCP server code, base address, SDA–18
MSCP symbol, SDA–18
Multiplication operator (*), SDA–16
Multiprocessing, global symbols, SDA–64
Multiprocessors
  analyzing crash dumps, SDA–12
  displaying synchronization structures, SDA–169

N

NAMs (name blocks), SDA–79
Negative operator (–), SDA–16
NETDEF.STB file, SDA–63
nnDRIVER symbol, SDA–18
/NODE qualifier, SDA–85, SDA–90
/NOLOGICAL_NAMES qualifier, SDA–183
Nonpaged dynamic storage pool, displaying contents, SDA–135
/NONPAGED qualifier, SDA–135
/NOSKIP qualifier, SDA–54
/NOSUPPRESS qualifier, SDA–54
/NOSYMBOLS qualifier, SDA–183
/NOTIFY qualifier, SDA–183
NOT operator (#), SDA–16
/NOWAIT qualifier, SDA–183
NWAs (network work areas), SDA–79

O

OpenVMS RMS
  See RMS
Operators
  precedence of, SDA–16, SDA–17
ORB symbol, SDA–18
OR operator (|), SDA–16
/OUTPUT qualifier, SDA–183

P

PFNs (page frame numbers)
P0BR register, displaying, SDA–95
P0BR symbol, SDA–18
P0LR register, displaying, SDA–95
P0LR symbol, SDA–18
P0 page table, displaying, SDA–150
/P0 qualifier, SDA–150
P0 region, examining, SDA–54
P1BR register, displaying, SDA–95
P1BR symbol, SDA–18
P1LR register, displaying, SDA–95
P1LR symbol, SDA–18
P1 page table, displaying, SDA–150
/P1 qualifier, SDA–54, SDA–150
P1 region, examining, SDA–54
Paged dynamic storage pool, displaying contents, SDA–135
/PAGED qualifier, SDA–135
Page faults, illegal, SDA–23
Page files
  See SYS$SYSTEM:PAGEFILE.SYS file
  using as system dump file, SDA–8
Page tables
  displaying, SDA–150
Page tables, displaying, SDA–126
PAGE_MANAGEMENT.EXE file, global symbols, SDA–64
/PAGE_TABLES qualifier, SDA–150
Parentheses (), as precedence operators, SDA–17
/PARENT qualifier, SDA–45
/PARTICIPANTS qualifier, SDA–150
PBs (path blocks), SDA–104
PCBB register, displaying, SDA–95
/PCB qualifier, SDA–150
PCBs (process control blocks), SDA–180
  displaying, SDA–150, SDA–151
  hardware, SDA–153

PCB symbol, SDA–18
PCs (program counters), SDA–18
PCs (program counters), in a crash dump, SDA–19
PC symbol, SDA–18
PDTs (port descriptor tables), SDA–142
PDT symbol, SDA–18
PFN database, SDA–126
PFN database, displaying, SDA–131
PGFIPLHI bugcheck, SDA–23
/PHD qualifier, SDA–150
PHDs (process headers), SDA–180
PHDs (process headers), displaying, SDA–150
PHD symbol, SDA–18
PID numbers
  SDA uses to extract correct index, SDA–149
Pool lists
  displaying contents, SDA–135
  statistics about, SDA–135
Port drivers, displaying SDA information, SDA–85
Ports, displaying SDA information, SDA–142
Positive operator (+), SDA–16
Precedence operators, parentheses used as, SDA–17
PRIMITIVE_IO.EXE file, global symbols, SDA–64
Process contexts, changing, SDA–71, SDA–76, SDA–98, SDA–149
Process control region, SDA–18
Process control region, operatior (H), SDA–16
Processes
  channel, SDA–149
  displaying
    SDA information, SDA–149, SDA–178
  examining hung, SDA–11
  image, SDA–178
  listening, SDA–86
  lock, SDA–150
  scheduling state, SDA–153, SDA–179
  spawning a subprocess, SDA–183
Process indexes, SDA–149
Process names, SDA–149
Processor context, changing, SDA–71, SDA–77, SDA–94, SDA–98, SDA–149
Processor-specific loadable code, base address, SDA–18
Processor status longwords
  See PSLs
Processor types, displaying, SDA–95
Process-permanent I/O structures, SDA–80
/PROCESS qualifier, SDA–184
PROCESS_MANAGEMENT.EXE file, global symbols, SDA–64
/PROCESS_SECTION_TABLE qualifier, SDA–150
Program regions
  base register, SDA–18
  displaying page tables, SDA–150
  examining, SDA–54
Program regions (cont'd)
  length register, SDA–18
/PSL qualifier, SDA–54
PSLs (processor status longwords)
  evaluating, SDA–26, SDA–51
  examining, SDA–54
  symbol, SDA–18
/PST qualifier, SDA–150
PSTs (process section tables) displaying, SDA–150
/PTE qualifier, SDA–51, SDA–54
PTEs (page table entries)
  evaluating, SDA–51
  examining, SDA–54
2P_CDDB symbol, SDA–17
2P_UCB symbol, SDA–17

Q

Queues
  stepping through, SDA–67
  validating, SDA–185

R

RABs (record access blocks), SDA–80
Radixes, default, SDA–16
Radix operators, SDA–16
RDTs (response descriptor tables), SDA–167
READ/EXECUTIVE command, SDA–20
READ command, SDA–62
READ command, SYS$DISK, SDA–63
Recovery unit system services, global symbols, SDA–64
RECOVERY_UNIT_SERVICES.EXE file, global symbols, SDA–64
Registers
  displaying, SDA–94, SDA–150
  general, SDA–18
/REGISTERS qualifier, SDA–150
/RELEASE qualifier, SDA–5
/RELOCATE qualifier, SDA–62
REPEAT command, SDA–67
Report system event, global symbols, SDA–64
Resources, displaying SDA information, SDA–161
Ring buffer, nonpaged pool history, SDA–135
/RING_BUFFER qualifier, SDA–135
RLBs (record lock blocks), SDA–80
RMS
  data structures shown by SDA, SDA–79
  displaying data structures, SDA–151, SDA–166
  global symbols, SDA–63, SDA–64
  image
    base address, SDA–18
  symbol, SDA–18
RMS.EXE file, SDA–64
RMSDEF.STB file, SDA–63

/RMS qualifier, SDA–151
RSBs (resource blocks), SDA–122, SDA–161
RSPID (response ID), displaying SDA information, SDA–167
RUBs (recovery unit blocks), SDA–80
RUFBs (recovery unit file blocks), SDA–80
RUSBs (recovery unit stream blocks), SDA–80
RWAITCNT symbol, SDA–18

S

S0 region, examining, SDA–54
SAVEDUMP system parameter, SDA–5
SBR register, displaying, SDA–95
SBs (system blocks), SDA–86, SDA–104
SB symbol, SDA–18
SCBB register, displaying, SDA–95
Schedulers, global symbols, SDA–64
SCS (System Communications Services)
  base address, SDA–18
  displaying SDA information, SDA–85, SDA–86, SDA–90, SDA–142, SDA–167
  global symbols, SDA–63
SCSDEF.STB file, SDA–63
SCSLOA symbol, SDA–18
/SCS qualifier, SDA–85
SDA$INIT logical name, SDA–10
SDA current CPU
  changing, SDA–14
  displaying, SDA–176
  implicitly setting using /SYSTEM qualifier, SDA–149
  implicitly setting using SHOW CRASH command, SDA–98
  selecting using SET CPU command, SDA–71
  selecting using SET PROCESS command, SDA–77
  using the SHOW CPU command, SDA–94
SDA current process
  changing, SDA–12
  changing using SHOW CRASH command, SDA–98
  displaying, SDA–176
  implicitly changed, SDA–14, SDA–71
  implicitly setting using /SYSTEM qualifier, SDA–149
  selecting using SET PROCESS command, SDA–76
SDA symbol table, SDA–17
  building, SDA–10
  expanding, SDA–10
SEARCH command, SDA–69
SECURITY.EXE file, global symbols, SDA–64
Self-relative queue, validating, SDA–185
/SELF_RELATIVE qualifier, SDA–185
SET CPU command, SDA–14, SDA–71
SET CPU command, analyzing a running system, SDA–11
SET LOG command, SDA–74
SET LOG command, compared with SET OUTPUT command, SDA–74
SET NOLOG command, SDA–74
SET OUTPUT command, SDA–75
SET OUTPUT command, compared with SET LOG command, SDA–74
SET PROCESS command, SDA–12, SDA–76
SET RMS command, SDA–79
/SET_STATE qualifier, SDA–48
SFSBs (shared file synchronization blocks), SDA–80
Shadow sets, displaying SDA information, SDA–105
Shifting operator (@), SDA–17
SHOW CALL_FRAME command, SDA–68, SDA–82
SHOW CLUSTER command, SDA–85
SHOW CONNECTIONS command, SDA–90
SHOW CPU command, SDA–14, SDA–71, SDA–94
  analyzing a running system, SDA–11
SHOW CRASH command, SDA–14, SDA–19, SDA–21, SDA–71, SDA–98
SHOW CRASH command, analyzing a running system, SDA–11
SHOW DEVICE command, SDA–20, SDA–27, SDA–103
SHOW EXECUTIVE command, SDA–20, SDA–110
SHOW HEADER command, SDA–112
SHOW LAN command, SDA–113
SHOW LOCK command, SDA–121
SHOW LOGS command, SDA–125
SHOW MEMORY command, SDA–4
SHOW PAGE_TABLE command, SDA–26, SDA–126
SHOW PFN_DATA command, SDA–131
SHOW POOL command, SDA–135
SHOW PORTS command, SDA–142
SHOW PROCESS/ALL command, SDA–152
SHOW PROCESS/LOCKS command, SDA–121
SHOW PROCESS/RMS command, SDA–166
SHOW PROCESS/RMS command, selecting display options, SDA–80
SHOW PROCESS command, SDA–77, SDA–149
SHOW RESOURCE command, SDA–121, SDA–161
SHOW RMS command, SDA–166
SHOW RSPID command, SDA–167
SHOW SPINLOCKS command, SDA–170
SHOW STACK command, SDA–25, SDA–176
SHOW SUMMARY command, SDA–149, SDA–178

SHOW SYMBOL command, SDA–181
SHOW TRANSACTIONS command, SDA–182
Shutdown, operator-requested, SDA–7
SID register, displaying, SDA–95
Signal array, SDA–22
SISR register, displaying, SDA–95
Site-specific startup procedure
  See SYS$MANAGER:SYSTARTUP_VMS.COM
SLR register, displaying, SDA–95
SPAWN command, SDA–183
Spin locks
  displaying SDA information, SDA–169
  owned, SDA–95
SPRs (Software Performance Reports), SDA–3, SDA–31
SP symbol, SDA–18
SPTs (system page tables)
  displaying, SDA–26, SDA–126
  in system dump file, SDA–4, SDA–9
SSP symbol, SDA–18
SSRVEXCEPT bugcheck, SDA–21
Stack frames
  displaying in SDA, SDA–82
  following a chain, SDA–82
Stack pointer, SDA–18
Stacks, displaying contents, SDA–176
Start I/O routine, SDA–104
/STATISTICS qualifier, SDA–135
Subprocesses, SDA–183
Subtraction operator (–), SDA–16
/SUMMARY qualifier, SDA–136
/SUPERVISOR qualifier, SDA–176
Supervisor stack
  displaying contents, SDA–176
  pointer to, SDA–18
Swapper, global symbols, SDA–65
Symbols, SDA–17 to SDA–19
  defining
    for SDA, SDA–47
  displaying, SDA–19
  evaluating, SDA–181
  finding in memory location, SDA–27
  listing, SDA–181
  loading into the SDA symbol table, SDA–62
  name, SDA–17, SDA–47
  representing executive modules, SDA–110
  user-defined, SDA–47
SYMBOLS qualifier, for SDA EVALUATE command, SDA–51
Symbol table files, reading into SDA symbol table, SDA–62
Symbol tables
  See SDA symbol table
  See system symbol table
  specifying an alternate SDA, SDA–39
SYS$DISK logical name, SDA–63
SYS$MANAGER:SYSTARTUP_VMS.COM command procedure
  invoking SDA, SDA–7
  producing an SDA listing, SDA–7
  releasing page file blocks, SDA–5
SYS$SYSTEM:OPCCRASH.COM command procedure
  involvement in writing crash dump, SDA–7
SYS$SYSTEM:PAGEFILE.SYS file, SDA–8, SDA–32
  See System dump files
  as dump file, SDA–5
  releasing blocks containing a crash dump, SDA–38
SYS$SYSTEM:REQSYSDEF.STB file, SDA–8, SDA–10
SYS$SYSTEM:SHUTDOWN.COM command procedure, involvement in writing crash dump, SDA–7
SYS$SYSTEM:SYS.EXE file, SDA–62
SYS$SYSTEM:SYS.EXE file, contents, SDA–63, SDA–110
SYS$SYSTEM:SYS.STB file, SDA–8, SDA–10, SDA–11, SDA–20
SYS$SYSTEM:SYSDEF.STB file, SDA–10
SYS$SYSTEM:SYSDUMP.DMP file, SDA–32
  See System dump files
  protection, SDA–7
  size of, SDA–4
SYSAP (system application), SDA–167
/SYSAP qualifier, SDA–90
SYSDEVICE.EXE file, global symbols, SDA–64
SYSGETSYI.EXE file, global symbols, SDA–64
SYSLICENSE.EXE file, global symbols, SDA–64
SYSLOA symbol, SDA–18
SYSMSG.EXE file, global symbols, SDA–64
System Dump Analyzer utility (SDA)
  commands, SDA–15 to SDA–19
  exiting, SDA–57
System dump files, SDA–4 to SDA–6
  copying, SDA–6
  header, SDA–7
  mapping physical memory to, SDA–9
  requirements for analysis, SDA–8
  saving, SDA–6
  size, SDA–4
System failures
  analyzing, SDA–19 to SDA–31
  causing, SDA–31 to SDA–35
  diagnosing from PC contents, SDA–19
  example, SDA–24 to SDA–31
  summary, SDA–98
System hang, SDA–31
System images
  contents, SDA–63, SDA–110
  global symbols, SDA–62

System management, creating a crash dump file, SDA–4
System map, SDA–20
System message routines, global symbols, SDA–64
System page file
  as dump file, SDA–5
  releasing blocks containing a crash dump, SDA–38
System page tables
  See SPTs
System processes, SDA–76
/SYSTEM qualifier, SDA–54, SDA–76, SDA–126, SDA–131, SDA–151
System region, examining, SDA–54
Systems
  analyzing running, SDA–3, SDA–11, SDA–35
  investigating performance problems, SDA–11
System space base address, SDA–18
System space operator (G), SDA–16
System symbol table, SDA–8, SDA–17
System time quadword, examining, SDA–54
SYSTEM_PRIMITIVES.EXE file, global symbols, SDA–64
SYSTEM_SYNCHRONIZATION.EXE file, global symbols, SDA–64

T

TCPIP$BGDRIVER.STB, global symbols, SDA–64
TCPIP$INTEETACP.STB, global symbols, SDA–64
TCPIP$INTERNET_SERVICES.STB, global symbols, SDA–64
TCPIP$NET_GLOBALS.STB file, SDA–63
TCPIP$NFS_GLOBALS.STB file, SDA–63
TCPIP$NFS_SERVICES.STB file, SDA–65
TCPIP$PROXY_GLOBALS.STB file, SDA–63
TCPIP$PROXY_SERVICES.STB file, SDA–65
TCPIP$PWIPACP.STB, global symbols, SDA–65
TCPIP$PWIPDRIVER.STB, global symbols, SDA–65
TCPIP$PWIP_GLOBALS.STB file, SDA–63
TCPIP$TNDRIVER.STB, global symbols, SDA–65
TCPIP$TN_GLOBALS.STB file, SDA–63
Terminal keys, defining for SDA, SDA–47
/TERMINATE qualifier, SDA–49
/TIME qualifier, SDA–54
TMSCP server code, base address, SDA–18
TMSCP symbol, SDA–18
/TRANSACTIONS qualifier, SDA–151
/TYPE qualifier, SDA–58, SDA–136

U

UCBs (unit control blocks), SDA–90
UCB symbol, SDA–18
Unary operators, SDA–16
/USER qualifier, SDA–176
User stacks
  displaying contents, SDA–176
  pointer, SDA–19
USP symbol, SDA–19

V

VALIDATE QUEUE command, SDA–185
VAXcluster environments
  base address of loadable code, SDA–17
  displaying SDA information, SDA–85
  summary display, SDA–85
VCBs (volume control blocks), SDA–105
VCB symbol, SDA–19
/VC qualifier, SDA–142
/VECTOR_REGS qualifier, SDA–152
Virtual address operator (@), SDA–16
Virtual address space, sufficient for system dump analysis, SDA–8
VIRTUALPAGECNT system parameter, SDA–8

W

WCBs (window control blocks), SDA–80
Working set lists, displaying, SDA–152
/WORKING_SET qualifier, SDA–152
WORKING_SET_MANAGEMENT.EXE file, global symbols, SDA–65
/WSL qualifier, SDA–152

X

XABs (extended attribute blocks), SDA–80
XOR operator (\), SDA–16
XQP (extended QIO processor), SDA–104