VAX/VMS System Dump Analyzer Reference Manual

Order Number: AA–Z429C–TE

April 1986

This document explains the use of the System Dump Analyzer (SDA) to analyze the running system and dumps of system failures.

Revision/Update Information: This document supersedes the *VAX/VMS System Dump Analyzer Reference Manual* Version 4.2.

Software Version: VAX/VMS Version 4.4

digital equipment corporation
maynard, massachusetts

April 1986

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital

ZK–3031

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.
In New Hampshire, Alaska, and Hawaii call 603-884-6660.
In Canada call 800-267-6215.
*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).
Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a registered trademark of the American Mathematical Society.

System Dump Analyzer Contents

FIGURES

Preface

Intended Audience

This document is for users who need to debug device drivers or other system code.

Structure of This Document

This document is composed of three major sections.

The Format Section is an overview of SDA and is intended as a quick reference guide. The format summary contains the DCL commands that invoke SDA, listing all qualifiers and parameters. The usage summary describes invoking and exiting from SDA, how to direct output, and any restrictions you should be aware of. The command summary lists all of the SDA commands.

The Description Section explains how to use SDA.

The Commands Section describes each of the SDA commands and the qualifiers and parameters used with each. The commands appear in alphabetical order.

Conventions Used in This Document

Convention	Meaning
RET	A symbol with a 1- to 3-character abbreviation indicates that you press a key on the terminal, for example, RET .
ctrl x	The phrase CTRL/x indicates that you must press the key labeled CTRL while you simultaneously press another key, for example, CTRL/C, CTRL/Y, CTRL/O.
$ SHOW TIME 05-JUN-1985 11:55:22	The command examples show all output lines or prompting characters that the system prints or displays in black letters. All user-entered commands are shown in red letters.
$ TYPE MYFILE.DAT	A vertical series of periods, or vertical ellipsis, means either that not all the data normally displayed by the system in response to the particular command is shown or that not all the data a user would enter is shown.
file-spec,...	A horizontal ellipsis indicates that additional parameters, values, or information can be entered.

Convention	Meaning
[logical-name]	The square brackets indicate that the enclosed item is optional. (Square brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks refers refer to double quotation marks (″). The term apostrophe (′) refers to a single quotation mark.

New and Changed Features

The following new commands have been added to the System Dump Analyzer:

- ATTACH
- SPAWN

Also, the following new qualifiers are available for the EVALUATE, EXAMINE, and SEARCH commands:

- EVALUATE /PSL
- EVALUATE /PTE
- EVALUATE /SYMBOLS
- EXAMINE /NOSUPPRESS
- EXAMINE /PTE
- SEARCH /LENGTH=length_specifier
- SEARCH /STEPS=step_factor

The ATTACH command allows you to switch control of your terminal to another process in your job. The /PARENT qualifier allows you to switch control of your terminal to the parent process of the current process.

The SPAWN command creates a subprocess from the current process. The context is copied from the current process to the spawned process.

The EVALUATE/PSL command evaluates the specified expression in the format of a processor status longword.

The EVALUATE/PTE command 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.

The EVALUATE/SYMBOLS command specifies that all symbols that are known to be equal to the evaluated expression are to be displayed.

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

The SEARCH/LENGTH command specifies the size of the expression value to be used for successful matching during searches of memory. The possible values of this qualifier are: BYTE, WORD, and LONGWORD.

The SEARCH/STEPS command controls the granularity of searching through the specified memory range. As each comparison of memory occurs, the value of this qualifier determines the next memory location to be searched. The possible step_factors are: BYTE, WORD, LONGWORD, and QUADWORD.

Note that the COPY command releases the dump pages in the paging file so that they are available for system paging. Note that once the COPY command has released the dump pages for paging use, the dump information in these pages may be lost. Subsequent dump analysis should be carried out on the copy of the dump file that was specified in the COPY command.

Logical operators have been added to the arithmetic operators in Section 6.2.2. They are the logical AND, logical OR, logical XOR, and logical NOT.

The SET PROCESS and SHOW PROCESS commands can now include quoted strings in the process name in addition to the previous capital letters, numbers, dollar sign ($), and underscore (_) characters.

The SHOW DEVICE command examples have been changed and now include shadow devices.

The SHOW CRASH command register list now includes the system identification register.

The SHOW PROCESS /RMS=IFAB display has been added to show the changes to that display.

Other minor changes were made to correct typographical errors or slight omissions.

System Dump Analyzer

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

FORMAT

ANALYZE/*qualifier file-spec[/SYMBOL=symbol-table]*

Command Qualifiers	Defaults
/CRASH_DUMP	*None.*
/RELEASE	*None.*
/SYSTEM	*None.*
/SYMBOL=	*SYS$SYSTEM:SYS.STB*

Command Parameter

file-spec

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

If the /RELEASE qualifier is specified, SDA does not allow you to analyze the specified dump file. The /RELEASE qualifier must be specified with the /CRASH_DUMP qualifier and is useful only when the system paging file is being used as a dump file. The /RELEASE qualifier has the effect of releasing the blocks in the paging file that were used to store the dump. This effectively and immediately deletes the dump file from the system paging file.

usage summary

Invoking

To analyze a system dump, issue the ANALYZE/CRASH_DUMP command. This causes SDA to read a dump file. If you do not specify the name of a dump file, SDA prompts you for it.

To analyze the running system, issue the ANALYZE/SYSTEM command. Do not specify a dump file when you use this qualifier.

To specify a symbol table to use in place of the default, use the /SYMBOL qualifier.

Exiting

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

Directing Output

Use the SET OUTPUT command to send all output from SDA to a file. You must supply the name of the file as a parameter to the SET OUTPUT command. The file produced is 132 columns wide and is formatted for output to a printer.

To redirect the output to your terminal, use the SET OUTPUT SYS$OUTPUT command.

Use the SET LOG command to send a copy of all the commands you type and all the output those commands produce to a file. You must supply the name of the file as a parameter to the SET LOG command. The file produced is 132 columns wide and is formatted for output to a printer.

Privileges/Restrictions
To examine the running system, your process must have Change-Mode-to-Kernel (CMKRNL) privilege. The CMKRNL privilege is needed to release the page file dump blocks when using either the COPY command or specifying the /RELEASE qualifier in the ANALYZE/CRASH_DUMP command.

To use SDA to analyze a dump, your process must have the privileges necessary for reading the dump file. This usually requires system privilege (SYSPRV), but your system manager can, if neccessary, allow less privileged processes to read the dump files.

commands

Syntax
SDA> command [/qualifier[,...]] [parameter] [/qualifier[,...]]

System Dump Analyzer Commands

@ (Execute Procedure)
ATTACH
    /PARENT
COPY
DEFINE
    /ECHO
    /IF_STATE
    /KEY
    /SET_STATE=state_name
    /TERMINATE
EVALUATE
    /CONDITION_VALUE
    /PSL
    /PTE
    /SYMBOLS
EXAMINE
    /ALL
    /CONDITION_VALUE
    /INSTRUCTION
    /NOSKIP
    /NOSUPPRESS
    /P0
    /P1
    /PSL
    /PTE
    /SYSTEM
    /TIME
EXIT
FORMAT
    /TYPE=block_type
HELP
READ
    /RELOCATE=expression
REPEAT
SEARCH
    /LENGTH=length_specifier
    /STEPS=step_factor
SET LOG

```
SET OUTPUT
SET PROCESS
    /INDEX=index_value
    /SYSTEM
SET RMS
SHOW CLUSTER
    /CSID=n
    /SCS
SHOW CONNECTIONS
    /ADDRESS=n
SHOW CRASH
SHOW DEVICE
    /ADDRESS=n
SHOW HEADER
SHOW LOCK
    /ALL
SHOW PAGE_TABLE
    /GLOBAL
    /SYSTEM
    /ALL
SHOW PFN_DATA
    /ALL
    /BAD
    /FREE
    /MODIFIED
    /SYSTEM
SHOW POOL
    /ALL
    /FREE
    /HEADER
    /IRP
    /LRP
    /NONPAGED
    /PAGED
    /SRP
    /SUMMARY
    /TYPE=block_type
SHOW PORTS
    /ADDRESS=n
SHOW PROCESS
    /ALL
    /CHANNEL
    /INDEX=nn
    /LOCKS
    /P0
    /P1
    /PAGE_TABLES
    /PCB
    /PHD
    /PROCESS_SECTION_TABLE
    /REGISTERS
    /RMS=option
    /SYSTEM
    /WORKING_SET
SHOW RESOURCE
    /ALL
    /LOCKID=nn
SHOW RMS
```

```
SHOW RSPID
    /CONNECTION=n
SHOW STACK
    /ALL
    /EXECUTIVE
    /INTERRUPT
    /KERNEL
    /SUPERVISOR
    /USER
SHOW SUMMARY
    /IMAGE
SHOW SYMBOL
    /ALL
SPAWN
    /INPUT=filespec
    /NOLOGICAL_NAMES
    /NOSYMBOLS
    /NOTIFY
    /NOWAIT
    /OUTPUT=filespec
    /PROCESS=process_name
VALIDATE QUEUE
    /SELF_RELATIVE
```

DESCRIPTION

The System Dump Analyzer is a utility that you can use to help determine the cause of system failures. To use this utility effectively, you must be familiar with VAX/VMS data structures.

This utility uses data in a crash dump file, a file that the system writes each time the system fails. After a failure, this file contains a copy of the contents of memory and a copy of the system's hardware context at the time of the failure. See the *VAX/VMS System Manager's Reference Manual* and the next section of this document for additional information regarding this file.

SDA performs the following operations:

- Assigns a value to a symbol
- Examines the memory of any process
- Formats instructions and blocks of data
- Displays data structures of devices
- Displays the RMS data structures of a process
- Displays memory management data structures
- Displays a summary of all processes on the system
- Displays the SDA symbol table
- Copies the system dump file
- Sends output to a file or device
- Reads global symbols from any object module
- Searches memory for a given value

In addition to analyzing the system's dump file, SDA can perform the operations listed previously on a running system without interrupting that system's operation.

1 Introduction to SDA

When a fatal error within the system interferes with normal operations by causing the system to fail, the VAX/VMS operating system writes information concerning its status to a system dump file. The System Dump Analyzer (SDA) reads, formats, and displays the contents of this file. You can use SDA to display information on a video display terminal or to create hardcopy listings.

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 DIGITAL a Software Performance Report (SPR) and a copy of the system dump file written at the time of the failure.

2 System Management and SDA

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

2.1 The System Dump File

The VAX/VMS operating system can write information into the system dump file only if the system parameter DUMPBUG is set. This parameter is set by default. To alter DUMPBUG, as well as other system parameters, consult the *VAX/VMS System Manager's Reference Manual.*

If the DUMPBUG parameter is set and the operating system fails, the system writes the contents of the error-log buffers, processor registers, and physical memory into the file SYS$SYSTEM:SYSDUMP.DMP, overwriting the contents of that file. SDA reads this file and produces formatted displays of its contents.

If the file SYSDUMP.DMP does not exist, the VAX/VMS 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 is retained in PAGEFILE.SYS when the system is booted. Otherwise, the entire paging file is used for paging and the dump in the paging file is lost. To save the dump and free the pages in the paging file taken up by the dump, this dump must be copied from the paging file to another file. Sections 2.3 and 2.4, and the COPY command description, indicate how to accomplish this.

Occasionally, you may want to free the pages in the paging file that are taken up by the dump without having to copy the dump elsewhere. By issuing the ANALYZE/CRASH_DUMP/RELEASE command, SDA immediately releases the pages to be used for system paging, effectively deleting the dump. Note that this command *does not* allow you to analyze the dump before deleting it.

System dump files are set to NOBACKUP, in a manner similar to files used for paging and swapping, which means that BACKUP will not copy them to tape unless you use the qualifier /IGNORE=NOBACKUP with BACKUP. When SDA copies the system's dump file to another file, it does not set the file to NOBACKUP.

2.2 Setting the Size of the Dump File

The file SYSDUMP.DMP is furnished in the VAX/VMS software distribution kit as an empty file located in SYS$SYSTEM, the system directory. The file SYSDUMP.DMP is small. You must make it large enough to hold all the information to be written when the system fails, or you must have the system dumps written into the paging file.

To calculate the right size for your system's dump file, SYS$SYSTEM:SYSDUMP.DMP, use the following equation:

```
Size-of-dump-file = size-of-physical-memory + 4
```

The size of the dump file is expressed in blocks, and the size of memory is expressed in pages. The four extra pages are used to save the error-log buffers and bugcheck information. Be sure to include any shared memory when figuring the size of your system's physical memory.

Use the SYSGEN utility to set the size of the dump file. See the *VAX/VMS System Generation Utility Reference Manual* for more information on establishing the size of dump files.

If you want to use the paging file as the dump file, you must use SYSGEN to set the system parameter SAVEDUMP and to set the size of the paging file. Determine its minimum size according to the following equation:

```
Size-of-paging-file = size-of-physical-memory + 4 + 1000
```

Note that this is a minimum for saving a dump. The paging file must be larger than this for most systems to avoid hanging the system. See the *VAX/VMS System Manager's Reference Manual* for more information.

2.3 Saving System Dump Files

Every time the operating system writes information to SYSDUMP.DMP, it writes over whatever was previously stored in the file. For this reason, the system manager should save the contents of SYSDUMP.DMP after a system failure has occurred.

The system manager can use the SDA COPY command or the DCL COPY command. Either command can be used in your site-specific startup procedure, but the SDA COPY command is preferred because it marks the dump file as copied. This is particularly important if the dump was written into the paging file, PAGEFILE.SYS. Section 2.4 discusses the startup procedure in more detail. See the Commands Section for a description of the COPY command.

2.4 The System Startup 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 make sure that SDA produces such a listing after every failure. The system manager can ensure the creation of a listing by modifying the SYSTARTUP.COM file in the SYS$MANAGER directory so that it invokes SDA when the system is booted.

When called by the system startup procedure, SDA executes the commands in the system's startup command procedure only if the system just failed. SDA scans the dump file for a flag that indicates whether SDA has already processed the file. This flag is cleared each time the operating system writes a crash dump into SYSDUMP.DMP (unless an operator requested shutdown with OPCCRASH.EXE). If the flag is cleared, SDA executes the startup command procedure and sets the flag. If SDA finds that the flag is set, however, it exits without executing the procedure.

The following example shows typical commands that might be added to your site-specific startup command file 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 LPAO:SYSDUMP.LIS        ! Create listing file
   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
```

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

Note that if you use the paging file, SYS$SYSTEM:PAGEFILE.SYS, as the crash dump file, you must use SDA to copy the dump from this file to another file. If you fail to do this, the pages in the paging file that were used to save information on the failure are not freed for use in paging, and your system might hang during the execution of STARTUP.COM.

Thus, if you use the paging file as the crash dump file, you must include the following commands in your system's STARTUP.COM file:

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

Various SDA commands

```
    .
    .
    .


SDA> COPY filespec
SDA> EXIT
```

For another method of releasing the pages in the paging file without analyzing the dump, see Section 2.1.

3 Using SDA

You can use SDA to examine the running system or the dump resulting from a system failure. The next two sections describe these activities.

When you invoke SDA, by using DCL's ANALYZE/SYSTEM command or by using DCL's ANALYZE/CRASH_DUMP command, SDA executes the commands in the SDA initialization file, if such a file exists. SDA refers to its initialization file by using the logical name SDA$INIT. This initialization file can contain SDA commands that define keys, among other SDA commands.

3.1 Analyzing a System Dump

To enable SDA to read a dump file, your process must have:

- Read access to the file that contains the dump
- Read access to a copy of the system symbol table, SYS.STB, which SDA reads by default
- Enough virtual address space for SDA to map the entire dump file, to map any symbol tables required, and to use for stacks

The files SYSDUMP.DMP and SYS.STB are included in the VAX/VMS distribution kit. World access is denied to the SYSDUMP.DMP file. Because the dump file can contain privileged information, it is a good idea for the system manager to protect SYSDUMP.DMP from universal read access. See the *VAX/VMS System Manager's Reference Manual* for details on how to change a file's protection.

To ensure that SDA has the correct amount of virtual address space, the value of the system parameter VIRTUALPAGECNT must be larger than the size of the system's dump file by approximately 2000 pages.

The suggested parameter setting is a sufficient guideline for the majority of VAX/VMS installations. Further increases in the parameter may be required if your particular installation places extra heavy demands upon the virtual address space of the process.

For more information on process privileges, see the *VAX/VMS System Manager's Reference Manual*. For a description of system parameters, see the *VAX/VMS System Generation Utility Reference Manual*. For a description of the Authorize Utility, see the *VAX/VMS Authorize Utility Reference Manual*.

If the conditions listed previously are satisfied, you can invoke SDA to examine a dump file with the DCL command ANALYZE/CRASH_DUMP. If you do not specify a dump file with this command, SDA prompts you for the name of the file, as follows:

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

The default file specification is SYS$DISK:[default-dir]SYSDUMP.DMP, where SYS$DISK and [default-dir] represent, respectively, the disk and directory specified in your last SET DEFAULT command. (See Section 6 for more information about SDA command formats.)

4 Reading the System Dump File

When you invoke SDA to analyze the system dump file, SDA gathers, from the specified dump file, the information it needs to create its displays. Under certain conditions, some memory locations might not be saved in the system dump file.

For instance, if SYSDUMP.DMP is too small, the operating system cannot copy all of memory to the file 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 SPT is included in the dump in its entirety.

The SPT, which contains one entry for each page of system virtual address space, is the first data structure SDA looks for when it reads the system dump file. As long as SYSDUMP.DMP contains the SPT, SDA can map the contents of that file.

Only the contents of physical memory are saved when the system fails. Thus, if you use an SDA command to access a virtual address that has no corresponding physical address, SDA generates an error message:

```
%SDA-E-NOTINPHYS, xxxxxxxx : not in physical memory
```

In the preceding message, xxxxxxxx represents the virtual address that is not in physical memory.

Also, during halt/restart bugchecks, the contents of general registers are not preserved. If such a bugcheck occurs, SDA indicates in the SHOW CRASH display that the contents of the registers were destroyed.

5 Analyzing a Running System

Occasionally, VAX/VMS encounters an internal problem that hinders system performance without causing 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.

To examine the running system, your process needs change-mode-to-kernel (CMKRNL) privilege. (See the *VAX/VMS System Manager's Reference Manual* for a discussion of this and other privileges.) If your process has CMKRNL privilege, you can invoke SDA to examine the system with the following DCL command:

```
$ ANALYZE/SYSTEM
```

SDA automatically sets process context to that of your process. (For an explanation of process context, see the description of the SET PROCESS command in the Commands section of this manual.)

In analyzing the system dump file, SDA maps the entire file; but in analyzing the running system, SDA does not map the entire system. Instead, each time you give a command, SDA retrieves only the information it needs to process that command. If you reissue the command, SDA fetches the information again. In this way, SDA updates requested information to reflect the current state of the running system.

You can use the ANALYZE/SYSTEM command 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. The *VAX/VMS System Manager's Reference Manual* provides more information about scheduler states.

5.1 Building the SDA Symbol Table

After locating and reading the system dump file, SDA attempts to read the system's symbol table file. This file, named SYS.STB, contains the global symbols used by the VAX/VMS operating system.

SDA looks for SYS.STB in the system directory SYS$SYSTEM. Once SDA finds SYS.STB, it copies the file's contents to the SDA symbol table. If SDA cannot find the system symbol table file, it halts with a fatal error.

In addition to having SDA read symbols in SYS.STB, you might find it useful for SDA to read the symbols in SYSDEF.STB, which contains symbols that define many of the system's data structures, including those in the I/O database. You can cause SDA to read these symbols by using the READ command described in the Commands section of this manual.

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 a nonzero mutex count at the end of the execution of a system service.

```
VAX/VMS System dump analyzer
Dump taken on 15-Feb-1986  10:15:49.20
MTXCNTNONZ, Mutex count nonzero at system service exit
SDA>
```

The SDA> prompt indicates that you can use SDA interactively and enter SDA commands, or send selected information to a file, or print selected information on a printer. Refer to the description of the SET OUTPUT command in the Commands section of this manual for directions on defining output files, and to the description of the SET LOG command for directions on defining log files.

6 SDA Command Format

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

6.1 General Command Format

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

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

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

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

When the parameter is a file specification, the default device is SYS$DISK, the device specified in your most recent SET DEFAULT command. Likewise, the default directory is the directory specified in the most recent SET DEFAULT command. See the *VAX/VMS DCL Dictionary* for a description of the DCL command SET DEFAULT.

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

The **!comment** is text that comments upon the command. Such comments are useful for documenting SDA command procedures. However, the exclamation point (!) may be used within an expression in a command to indicate a logical OR operation. If the exclamation point (!) is used outside of the expression, SDA ignores the exclamation point and all characters that follow it on the same line (they are treated as a comment).

6.2 Expressions

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

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

Numerals are the digits you can type on your keyboard. The following sections describe the use of the other components of expressions.

6.2.1 Radix Operators

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

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

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

6.2.2 Arithmetic and Logical Operators

Arithmetic and logical operators are useful in forming expressions. There are two types: unary and binary operators. Unary operators affect the value of the expression that follows them. Binary operators combine the operands that precede and follow them. The SDA arithmetic operators perform integer arithmetic on 32-bit operands.

SDA recognizes the following six unary operators:

Operator	Action
#	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 to the value of the expression
H	Adds 7FFE0000 to the value of the expression

The unary operator G corresponds to the first virtual address in system space; the unary operator H corresponds to a convenient base address in the control region of a process. The binary operators are the following:

Operator	Action
+	Addition
–	Subtraction
*	Multiplication
&	Logical AND
!	Logical OR
\	Logical XOR
/	Division
@	Arithmetic shifting

SDA performs logical AND, OR, and XOR operations, and multiplication, division, and arithmetic shifting before addition and subtraction. In division, SDA truncates the quotient to an integer, if necessary, and does not retain a remainder. Note that the logical OR operator (!) is valid within the expression in a command that uses an arithmetic expression. Otherwise, in an SDA command, the DCL command line interpreter may interpret the exclamation point as the start of a comment on the command line.

6.2.3 Precedence Operators

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

6.2.4 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 to 7FFFFFFF (hexadecimal).

SDA copies symbols into its symbol table from the SYS.STB file. Additional symbols can be taken from other symbol tables and added to the SDA symbol table with the READ command. Symbols can also be created by the DEFINE command.

In addition, SDA provides the following symbols:

Symbol	Meaning
. (the period character)	The current location
AP	The argument pointer
CLUSTRLOA	The base address of loadable VAXcluster code
nnDRIVER	The base address of the driver prologue table (DPT); a symbol exists for each loaded device driver in the system
ESP	The executive-mode stack pointer
FP	The frame pointer
FPEMUL	The base address of the code that emulates floating-point instructions
G	80000000, the base address of system space
H	7FFE0000
KSP	The kernel-mode stack pointer
MCHK	An address within loadable CPU-specific routines
MP	The base address of loadable multiprocessor code
MSCP	The address of loadable MSCP-server code
P0BR	The base register for the program region
P0LR	The length register for the program region
P1BR	The base register for the control region
P1LR	The length register for the control region
PC	The program counter
PSL	The processor-status longword
R0 through R11	The general registers
RMS	The base address of the RMS image
SCSLOA	The base address of loadable common SCS services

Symbol	Meaning
SP	The current stack pointer of a process
SSP	The supervisor-mode stack pointer
SYSLOA	The base address of loadable processor-specific system code
USP	The user-mode stack pointer

The register symbols correspond to the registers saved as part of the hardware context of the current process. (See the SET PROCESS command in the Commands section of this manual for a definition of the current process.) For example, the command EXAMINE @USP displays the contents of the user-mode stack pointer, the register that contains the address of the user-mode stack.

The notation nn within the symbol nnDRIVER represents the 2-letter, generic device name (for example, LPDRIVER).

When SDA displays an address, it displays that address both in hexadecimal and as a symbol, if possible. If the address is within FFF 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 of the address, SDA displays the symbol whose value is the closest. If no symbols have values within FFF of the address, SDA displays no symbol. For an example, see the description of the SHOW STACK command.

7 Analyzing System Failures

The next sections discuss how the VAX/VMS operating system handles internal errors, and suggests procedures that can aid you in determining the cause of these errors. The last sections illustrate, through detailed analysis of a sample system failure, how SDA helps you find the cause of operating system problems.

For a complete description of the commands discussed in the sections that follow, refer to the last part of the SDA documentation, which describes all the SDA commands in alphabetical order.

7.1 General Procedure for Analyzing System Failures

When the VAX/VMS 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 bugcheck code describes the error; each error is associated with a code.

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 DIGITAL. 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 DIGITAL. A hardware failure requires the attention of Digital Field Service. To diagnose an error in code supplied by DIGITAL, you need listings of that code, which are supplied on microfiche with your VAX/VMS software kit.

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 content of the program counter (PC). The content of the PC is the address of the next instruction after the instruction that signaled the bugcheck.

The PC often contains an address in the exception handler, which signaled the bugcheck but did not cause it. In this case, the address of the instruction that caused the bugcheck is located on the stack. Use the SHOW STACK command to display the contents of the stack. See Section 7.2 for information on how to proceed for several types of bugchecks.

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

If it is not part of a driver, examine the linker's map of the module or modules you are debugging to determine if the instruction that caused the bugcheck is in your programs.

If it is not part of a driver and not part of your programs, examine the system map in the file SYS$SYSTEM:SYS.MAP. This file lists the addresses of each VAX/VMS module that is part of SYS.EXE, the system image. Compare the address in the PC with the addresses in the system map file to locate the module that contains the instruction to which the PC points.

If you do not have the system map file, you can use the SDA symbol table. All the global symbols that appear in SYS.MAP also exist in the file SYS.STB, which SDA reads when you invoke it. To determine the offset from the closest global symbol, you can issue the command

```
SDA> EXAMINE @PC
```

Note, however, that the closest symbol might not be in the same module as the code you are examining.

Once you have narrowed the search to a particular module, subtract the module's starting address from the address in the PC to get the offset into the module of the failing instruction.

Now, to determine the general cause of the system failure, examine the code that signaled the bugcheck.

7.2 Fatal Bugcheck Conditions

Bugchecks frequently are caused by one of two conditions: a fatal exception or an illegal page fault.

7.2.1 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, it lists the approximate cause of the failure on the console terminal as shown following. SDA displays the same information when you use the SHOW CRASH command. SDA displays one of the following reasons for a fatal exception:

```
FATALEXCPT, Fatal executive or kernel mode exception
INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
SSRVEXCEPT, Unexpected system service exception
```

For INVEXCEPTN and SSRVEXCEPT bugchecks, two lists of arguments are pushed on the stack, the signal array and the mechanism array. These lists contain important information about the exception that resulted in the bugcheck.

Although there are several possible exception conditions, access violations are most common. The rest of this section discusses access violations in detail. For more information on other kinds of exceptions and condition handling, see the *VAX/VMS Run-Time Library Routines Reference Manual.*

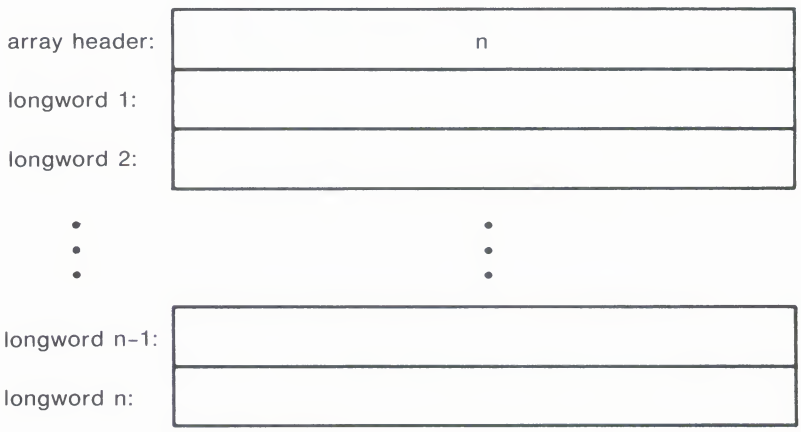
When the hardware detects an access violation, information useful in finding the cause of the violation is pushed onto the kernel-mode stack. If the access violation occurred when the system was using the interrupt stack, the argument lists go on the interrupt stack. This information is described by three arrays.

An argument list is a series of longwords, in which the first longword contains the number of longwords that follow. For an exception, the first argument list that appears on the stack contains the addresses of the next two arrays. In this way, the first argument list, or array, that may appear on the stack contains three longwords: the first contains the number of longwords that follow, the second is the address of a signal array, and the third is the address of a mechanism array. Thus, each longword following the first contains the address of the next two lists called the mechanism array and the signal array. The general form of an argument list or array is shown in Figure SDA-1. The first argument list on the stack is shown in Figure SDA-2.

The first longword in the mechanism array or the signal array contains the number of longwords that follow in each. The longwords in the mechanism array contain information that describes conditions at the time of the exception. These conditions are the stack frame, the stack depth, R0, and R1. The longwords in the signal array contain information that describes the exception code, exception parameters (if any), the program counter (PC), and the PSL. A mechanism array is shown in Figure SDA-3, and a signal array is shown in Figure SDA-4

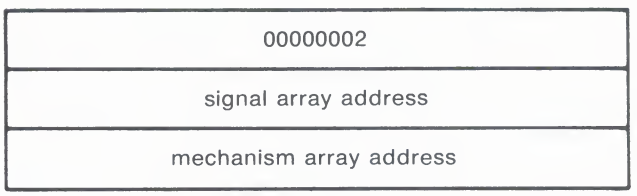
The first array does not always appear on the stack. If it does not, the mechanism array is the first. The mechanism array describes the process that was executing when the exception occurred. Figure SDA-3 illustrates the sequence of longwords in a mechanism array.

Figure SDA–1 An Argument List on the Stack



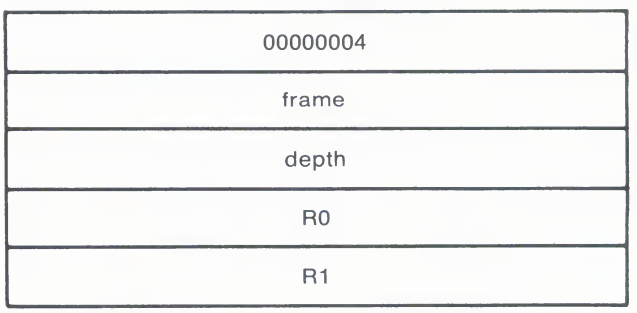
ZK-1919-84

Figure SDA–2 The First Argument List on the Stack



ZK-1920-84

Figure SDA–3 A Mechanism Array



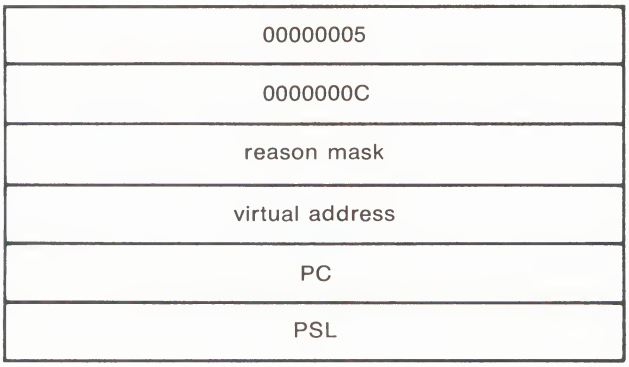
ZK-1921-84

The values contained in the mechanism array are defined as follows:

Value	Meaning
00000004	The number of longwords that follow. In a mechanism array, this value is always 4.
Frame	The address of the stack frame.
Depth	The stack depth.
R0	The contents of R0 at the time of the exception.
R1	The contents of R1 at the time of the exception.

The next argument list on the stack is the signal array. For access violations, the signal array is set up as follows:

Figure SDA–4 A Signal Array



ZK-1922-84

The values in the signal array are defined as follows:

Value	Meaning
00000005	The number of longwords that follow. For access violations, this value is always 5.
0000000C	The exception code. This value, C (hexadecimal), represents an access violation.
Reason mask	A longword, of which the lowest three bits, if set, indicate that the instruction caused a length violation (bit 0), referenced the process page table (bit 1), or attempted a read or modify operation (bit 2; read=0, write=1).
Virtual address	The virtual address that the system tried to reference.
PC	The program counter. The PC contains the address of the instruction that signaled the exception.
PSL	The processor status longword at the time of the exception.

Signal arrays differ in length, depending on the kind of exception the system detects. They contain a minimum of three arguments: the exception code, the PC, and the PSL at the time of the exception. You can identify the exception code by using the EVALUATE/CONDITION command.

If VAX/VMS encounters 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, 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 by the procedure outlined in Section 8.3.

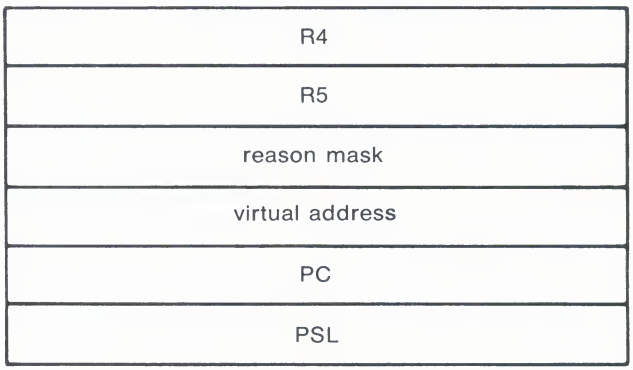
7.2.2 Illegal Page Faults

VAX/VMS also signals a bugcheck when a page fault occurs while the interrupt priority level (IPL) is greater than 2 (IPL$_ASTDEL). When VAX/VMS fails because of an illegal page fault, it prints the following message on the console terminal:

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

When an illegal page fault occurs, longwords containing the following information are pushed onto the operating stack.

Figure SDA–5 The Stack Following an Illegal Page-Fault Error



ZK-1923-84

The longwords pushed onto the stack are listed following.

Longword	Contents
R4	The contents of R4 at the time of the bugcheck.
R5	The contents of R5 at the time of the bugcheck.
Reason mask	A longword, of which the lowest three bits, if set, indicate that the instruction caused a length violation (bit 0), referenced the process page table (bit 1), or attempted a read or modify operation (bit 2; read=0, write=1).
Virtual address	The virtual address being referenced by the instruction that caused the page fault.
PC	The program counter. The PC contains the address of the instruction that was being executed when the page fault occurred.
PSL	The contents of the processor status longword at the time of the bugcheck.

If the operating system detects a page fault while the IPL is higher than 2, 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 8.3 to determine which module issued the instruction.

8 A Sample System Failure

This section steps through the analysis of a system failure caused by an example device driver. Three events lead up to this failure:

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

8.1 Identifying the Bugcheck

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

```
VAX/VMS System dump analyzer
Dump taken on 31-JAN-1985    16:34:31.23
INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
SDA>
```

VAX/VMS encountered an exception that caused it to signal a bugcheck. Signal and mechanism arrays are created on the current operating stack.

8.2 Identifying the Exception

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

```
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 and ends at address 8006A3B8. Its first longword contains 00000004. The signal array begins at address 8006A3C0 and ends at 8006A3D4. Its first longword contains 00000005 and its second longword contains 0000000C. Examination of the signal array shows that:

- The exception code is C (hexadecimal), which means that an access violation occurred.
- The reason mask is zero, which means that the instruction generated a protection violation (instead of a length violation) when it tried to read the location (rather than write to it).
- The virtual address is 80069E00, the address that the instruction tried to reference.
- The PC contains 8005D003, the address of the instruction that signaled the exception.
- 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 example following. The page containing location 80069E00 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
   .
   .
   .
```

8.3 Locating the Source of the Exception

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

8.3.1 Finding the Driver by Using the Program Counter

When SDA builds its internal symbol table, it defines a symbol, in the form nnDRIVER, for each device driver connected to the system. This symbol represents the base of the driver prologue table (DPT).

The DPT describes the driver. All of the driver prologue tables are linked in a list. Each DPT is part of the device driver it describes and is followed by that driver's code. The following example shows a partial list of the drivers in the display generated by the SHOW DEVICE command.

```
SDA> SHOW DEVICE
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
   .
   .
   .
```

Find the PC in the signal array, and examine that location in MACRO-instruction format by using the EXAMINE/INSTRUCTION command. If that address lies within the first 1000 (hexadecimal) bytes of a device driver, SDA identifies that driver by showing the address as a symbol, nnDRIVER, and an offset from that symbol to the address in question. The address defined as the symbol nnDRIVER is the base address of the driver's DPT.

If SDA is unable to find a symbol within 1000 (hexadecimal) bytes of the memory location you specify, it displays the location as an absolute address. This result often, but not always, means the instruction that caused the exception is not part of a device driver.

To determine whether an instruction is or is not part of a driver, use the SHOW DEVICE command (see the Commands section in this manual) to display the starting addresses and lengths of 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.

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

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

8.3.2 Calculating the Offset into the Driver's Program Section

The offsets that SDA displays are offsets from the DPT. These offsets do not match the offsets shown in driver listings. The offsets in a MACRO-assembler listing are offsets from the beginning of the program section (PSECT) in which that instruction appears. Because a driver usually contains more than one PSECT, you must use the driver's map to determine the following:

- The PSECT that contains the instruction
- The base address of that PSECT, relative to the base address of the DPT

To calculate the failing instruction's offset into the driver's program-section listing, subtract the PSECT base from the offset given by SDA.

8.4 Finding the Problem Within the Routine

To find the problem within the routine, examine the printer's driver code. In this sample system failure, 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 is stored in R3.

```
570 STARTIO:
571      MOVL    UCB$L_IRP(R5),R3      ;Retrieve address of I/O packet
572      MOVW    IRP$L_MEDIA+2(R3),-
573         UCB$W_BOFF(R5)             ;Set number of characters to print
574      MOVL    UCB$L_SVAPTE(R5),R3   ;Get address of system buffer
575      MOVAB   12(R3),R3             ;Get address of data area
576      MOVL    UCB$L_CRB(R5),R4      ;Get address of CRB
577      MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4 ;Get device CSR address
578 ;
579 ; START NEXT OUTPUT SEQUENCE
580 ;
582 10$: ADDL3   #LP_DBR,R4,R0         ;Find address of data buffer register
583      MOVZWL  UCB$W_BOFF(R5),R1     ;Get number of characters remaining
584      MOVW    #^X8080,R2            ;Get control register test mask
585      BRB     25$                   ;Start output
586 20$: BITW    R2,(R4)               ;Printer ready or have paper problem?
587      BLEQ    30$                   ;If LEQ not ready or paper problem,
588      MOVB    (R3)+,(R0)            ;output next character
589      FREEIB                        ;Flush instruction buffer and delay
590 25$: SOBGEQ  R1,20$                ;Any more characters to output?
591      BRW     70$                   ;All done, BRW to set return status
```

The instruction that caused the system failure is at line 588. The contents of R3 have probably been incremented too many times.

8.4.1 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 five steps:

1. The driver checks the printer's status register to see if the printer is ready.
2. 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; then it decrements R1, which contains the count of characters left to print. If R1 contains a number greater than zero, control is passed back to the instruction at 20$, and the loop begins again.
3. Steps 1 and 2 are repeated until the contents of R1 is 0 or the printer signals that it is not ready.
4. If the printer signals that it is not ready, the driver transfers control to 30$ (line 587), the beginning of a routine that waits for an interrupt from the printer.
5. When the printer is 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. This value controls the number of times the loop is executed. 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. 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.

Because the exception is an access violation, either R3 or R0 contains an incorrect value. Because the instruction at 10$ places the address of the data-buffer register into R0, and no other instruction references R0 until the MOVB instruction does, it is probable that R0 contains the correct value.

You can check whether R3 contains an incorrect value, however, by noting that the instruction at line 576 (MOVL UCB$L_CRB(R5),R4) moves the address of the printer's CRB into R4 and that no instruction changes the contents of R4 between lines 576 and 588. Although it is possible that the UCB contains the wrong address, 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 588 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.

8.4.2 Checking the Values of Key Variables

Because this 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 following.

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

8005D160    UCB$L_RQFL      800039A8
            UCB$L_FQFL
8005D164    UCB$L_RQBL      800039A8
            UCB$L_FQBL
8005D168    UCB$W_SIZE          0080
8005D16A    UCB$B_TYPE        10
8005D16B    UCB$B_FIPL      08
   .
   .
   .
8005D1C8    UCB$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 need not use the FORMAT command. Instead, you can use the command SHOW DEVICE LP. Because only one printer is connected to the VAX 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.

8.4.3 Identifying and Fixing 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 598).
- When the wait-for-interrupt (WFIKPCH) routine's timeout count of 12 seconds is exhausted (lines 603 and 615). At this time, the contents of R1, plus one is stored in UCB$W_BOFF (line 616).

When the printer times out, the driver should not modify UCB$W_BOFF. It does so, however, in line 616. The driver should modify the contents of UCB$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 WFIKPCH routine preserves only registers R3, R4, and R5, so only these registers can be used unmodified after the execution of the WFIKPCH routine. Thus the use of R1 in line 616 is an error.

To correct the problem, change the WFIKPCH argument (line 60) so that, when the printer times out, the WFIKPCH macro transfers control to 50$ rather than to 40$.

```
596
597 30$: BNEQ    40$                    ;If NEQ paper problem
598      ADDW3   #1,R1,UCB$W_BOFF(R5) ;Save number of characters remaining
599      DSBINT  UCB$B_DIPL(R5)        ;Disable interrupts
600      BITW    #^X80,LP_CSR(R4)      ;Is it ready now?
601      BNEQ    35$                   ;If NEQ, yes, it's ready
601      BISB    #^X40,LP_CSR(R4)      ;Set interrupt enable
603      WFIKPCH 40$,#12               ;Wait for ready interrupt
604      IOFORK                        ;Create a fork process
605      BRB     10$                   ;  ...and start next output
606
607 35$:
608      ENBINT                        ;Enable system interrupts
609      CLRW    LP_CSR(R4)            ;Disable device interrupts
610      BRB     10$                   ;Go transfer more characters
611 ;
612 ; PRINTER HAS PAPER PROBLEM
613 ;
614
615 40$: CLRL    UCB$L_LP_OFLCNT(R5)  ;Clear offline counter
616      ADDW3   #1,R1,UCB$W_BOFF(R5) ;Save number of characters remaining
617 50$: CLRW    LP_CSR(R4)            ;Disable printer interrupt
618      SETIPL  UCB$B_FIPL(R5)        ;Lower to fork level
619      TSTW    LP_CSR(R4)            ;Printer still have paper problem?
620      BLSS    55$                   ;If LSS yes
621      MOVL    #15,UCB$L_LP_TIMEOUT(R5)  ;Set timeout value
622      BRB     10$                   ; ...and start next output
```

8.5 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 at the end of this section.

Meeting Crash Dump Requirements

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

1. 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.

2. There must be a crash dump file in SYS$SYSTEM: named either SYSDUMP.DMP or PAGEFILE.SYS.

   If the dump file is SYSDUMP.DMP, it must be at least four blocks larger than physical memory.

If SYSDUMP.DMP is not present, VAX/VMS will write crash dumps to PAGEFILE.SYS. In this case, PAGEFILE.SYS must be at least 1004 blocks larger than physical memory, and the SYSBOOT parameter SAVEDUMP must be 1 (the default is 0).

3 The SYSBOOT DUMPBUG parameter must be 1 (the default is 1).

Examples of How to Cause System Failures

The following examples show the sequence of commands needed to cause a system failure on each type of VAX processor. On most processors, the console command file CRASH.COM or CRASH.CMD performs these steps for you.

EXAMPLES

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

The preceding example shows how to cause a system failure on a VAX 11/725 or a VAX 11/730. CTRL/P automatically halts the processor.

2
```
$ CTRL/P
>>> H
>>> E P
>>> E/I 0
>>> E/I +
>>> E/I +
>>> E/I +
>>> D/G F FFFFFFFF
>>> D P 1F0000
>>> C
```

The preceding example shows the steps needed to cause a system failure on a VAX 11/750. On these processors, the HALT command is a NOP; a CTRL/P automatically halts the processor.

3
```
$ CTRL/P
>>> HALT
HALTED AT 80008A89
>>> EXAMINE PSL
00000000
>>> EXAMINE/INTERN/NEXT:4 0
I  00000000  80008A89
I  00000001  00000000
I  00000002  00000000
I  00000003  00000000
I  00000004  80151E00
>>> DEPOSIT PC = -1
>>> DEPOSIT PSL = 1F0000
>>> CONTINUE
**** FATAL BUG CHECK, VERSION = X2M9 INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
CURRENT PROCESS = NULL
REGISTER DUMP
    R0 = 01F
    .
    .
```

.

The preceding example indicates how to cause a system failure on a VAX 11/782, VAX 11/785, or a VAX 11/780. Note that the value placed in the PC, 1F0000, sets the processor-access mode to kernel and the IPL to 31.

4
```
$ CTRL/P
        PC = 80008B1F
>>> D P 1F0000
>>> E P
        001F0000
>>> D/G F FFFFFFFF
>>> C
**** FATAL BUG CHECK, VERSION = 4.4 INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
    CURRENT PROCESS - NULL
    REGISTER DUMP
    .
    .
    .
```

The preceding example shows the steps needed to cause a system failure on a VAX 11/8200. On these processors, the HALT command is a NOP; a CTRL/P automatically halts the processor.

5
```
$ CTRL/P
>>> @CRASH
>>> SET QUIET OFF
>>> SET ABORT OFF
>>> HALT
        CPU stopped, INVOKED BY CONSOLE (CSM code 11)
        PC 80008B1F
>>> UNJAM
>>> E PSL
        U PSL 00000000
>>> E/I/N:4 0
        I 00 80000C40
        I 01 00000000
        I 02 00000000
        I 03 00000000
        I 04 00000000
>>> E SP
        G 0E 80000C40
>>> E/vir/next:40 @
        P 04206840 00000000
        P 04206844 00000000
    .
    .
    .
        P 0420693C 00000000
        P 04206940 00000000
>>> D PC FFFFFFFF
>>> D PSL 1F0000
>>> SET ABORT ON
>>> SET QUIET ON
**** FATAL BUG CHECK, VERSION = X4.4 INVEXCEPTN, Exception while above ASTDEL or on interrupt stack
    CURRENT PROCESS NULL
    REGISTER DUMP
    .
    .
    .
```

The preceding example shows how to cause a system failure on a VAX 8600 or VAX 8650.

6

```
$ CTRL/P
>>> SET CPU CURRENT_PRIMARY
>>> HALT
?00       Left CPU -- CPU halted
          PC = 8001911C
>>> @CRASH
!
! COMMAND PROCEDURE TO FORCE VMS BUGCHECK VIA ACCESS VIOLATION
!
SET VERIFY
SET CPU CURRENT_PRIMARY    !SELECT PRIMARY
EXAMINE PSL                !DISPLAY PSL
        M 00000000 00420008
EXAMINE/I/NEXT 4 0
        I 00000000 7FFE7DC8
        I 00000001 7FFE9518
        I 00000002 7FFEDE00
        I 00000003 0000F39C
        I 00000004 80873400
DEPOSIT PC FFFFFFFF        !SET PC=-1 TO FORCE ACCVIO
DEPOSIT PSL 41F0000        !SET IPL=31, INTERRUPT STACK
CONTINUE                   !EXECUTE FROM PC=-1
**** FATAL BUG CHECK, VERSION = X4.3 INVEXCEPTN, Exception while above ASTDEL
    CURRENT PROCESS = STARTUP
    REGISTER DUMP
        R0 = 0000001F
    .
    .
    .
```

The preceding example shows how to cause a system failure on a VAX 8800.

COMMANDS

The commands described in the following section can be used in analyzing a system dump or the running system.

@ (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

@file-spec

command parameter

file-spec

The 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 shown previously carries out the SDA commands contained in the file USUAL.COM, shown following.

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

This command procedure makes the file LASTCRASH.LIS the destination for output generated by subsequent SDA commands. Next, the command procedure sends information on the crash, the process, the stacks, and a summary of information to that file. Then it exits from the utility.

The procedure need not exit from the utility at the end its execution. To continue using SDA interactively after the execution of a command procedure, omit the EXIT command from the file.

ATTACH

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

FORMAT

ATTACH *process_name*

command parameter

process_name

The name of the process to which you want to transfer control.

command qualifier

/PARENT

This qualifier specifies that control of your terminal is to be switched to the parent process of your current process. If you specify this qualifier, do not specify the process_name parameter in the ATTACH command.

EXAMPLES

1
```
SDA> ATTACH/PARENT
```

The ATTACH command attaches the terminal to the process that is the parent of your current process.

2
```
SDA> ATTACH DUMPER
```

The ATTACH command attaches the terminal to the process that is named DUMPER.

COPY

Copies the contents of the dump file to another file.

FORMAT

COPY *output-file-spec*

command parameter

output-file-spec

The 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. You must supply the name of the file.

DESCRIPTION

Each time the system fails, the system copies all of physical memory and the hardware context of the current process into the file SYS$SYSTEM:SYSDUMP.DMP (or the paging file), overwriting the contents of that file. To preserve a crash dump, you must use the COPY command to copy the contents of this file into another file. The contents of SYSDUMP.DMP are not affected by execution of the COPY command.

The command procedure SYSTARTUP.COM should include this command to ensure that a copy of the dump file is made each time the system fails.

If the paging file was used as a dump file instead of SYSDUMP.DMP, the pages of the paging file that contain the dump information are not available for paging until they are explicitly released. The COPY command releases the dump pages in the paging file so that they are available for system paging if process privilege has been set to change-mode-to-kernel (CMKRNL). If CMKRNL privilege has not been set, the copy operation succeeds but the blocks used by the dump in the paging file are not released. Note that once the COPY command has released the dump pages for paging use, the dump information in these pages may be lost. Subsequent dump analysis should be carried out on the copy of the dump file that was specified in 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.

FORMAT

DEFINE*[/KEY] symbol [=] expression [/qualifier...]*

command parameters

symbol

The name you want to give the symbol. The symbol name can contain from 1 to 31 alphanumeric characters. See Section 6.2.5 for a discussion of SDA symbols.

If used with the /KEY qualifier, this parameter is the name of the terminal key to be defined. A list of the keys you can define, and their names, follows:

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

When you define some keys as SDA commands, you must press CTRL/V before those keys to execute the commands. This is due to the escape sequences these 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

expression

An expression that defines the value of the symbol. See Section 6.2 for a discussion of SDA expressions.

When you use the /KEY qualifier, this parameter is the SDA command the key is to be defined as.

command qualifiers

/[NO]ECHO

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.

/[NO]IF_STATE=(state-name,...)

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

/KEY

Causes a key, rather then a symbol, to be defined. If you use this qualifier, the **symbol** parameter must be the name of a key on your terminal keyboard, and the **expression** parameter must be the SDA command that is to be executed when the key, followed by carriage return, is pressed.

/SET_STATE=state-name

Causes the key being defined to cause a key state change rather than executing an SDA command. Instead of the name of a terminal key, the **expression** parameter must be the name of a key state. The key state is any name that you want to define. 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 the other keys, one in the **gold** state and one in the not-gold state. An example of this sort of multiple key definition is shown in example seven for this command.

/[NO]TERMINATE

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 the /TERMINATE qualifier is specified.

DESCRIPTION

SDA evaluates the expression, then assigns its value to the symbol. If the symbol is already defined, the new value replaces the old one. The symbol remains defined until you exit from SDA.

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 result of the computation but does not add symbols to the SDA symbol table.

EXAMPLES

1

```
SDA> DEFINE BEGIN = 80058E00
SDA> DEFINE END = 80058E60
SDA> EXAMINE BEGIN:END
```

In the preceding example, DEFINE defines two addresses, called BEGIN and END. These symbols serve as reference points in memory, defining a range of memory locations between which the EXAMINE command can examine.

2

```
SDA> DEFINE NEXT = @PC
SDA> EXAMINE/INSTRUCTION NEXT
00000454:   MOVL 6(R1), R3
```

Symbol NEXT defines the address contained in the program counter. SDA represents nonprinting characters by a period (.) and puts quotation marks around ASCII text. Refer to Section 6.2.5 for a discussion of SDA symbols.

3

```
SDA> DEFINE VEC SCH$GL_PCBVEC
```

This command assigns the value of a global symbol, SCH$GL_PCBVEC, to the symbol VEC. Now you can use the symbol VEC to access the memory location or value represented by the global symbol.

4

```
SDA> DEFINE COUNT = 4
SDA> DEFINE RESULT = COUNT * COUNT
SDA> EVALUATE RESULT
Hex =  00000010     Decimal =  16
```

The preceding example assigns symbol COUNT the value 4 and then uses the symbol in an arithmetic expression.

5

```
SDA> DEFINE/KEY PF1 "SHOW STACK"
SDA> PF1  SHOW STACK RETURN
Current operating stack
-----------------------
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 preceding example shows the DEFINE/KEY command being used to define PF1 as the SDA SHOW STACK command. When the PF1 key is pressed, SDA displays the command and waits for a carriage return to be typed.

6
```
SDA> DEFINE/KEY/TERMINATE PF1 "SHOW STACK"
SDA> PF1  SHOW STACK
Current operating stack
-----------------------
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 preceding example shows the DEFINE/KEY command being used to define PF1 as the SDA SHOW STACK command. The use of the /TERMINATE qualifier causes SDA to execute the SHOW STACK command without waiting for a carriage return to be typed.

7
```
SDA> DEFINE/KEY/SET_STATE="GREEN" PF1 ""
SDA> DEFINE/KEY/TERMINATE/IF_STATE=GREEN PF3 "SHOW STACK"
SDA> PF1 PF3  SHOW STACK
Current operating stack
-----------------------
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 preceding example shows the definition of two keys. PF1 is defined as a key that sets a command state GREEN. The trailing pair of quotation marks are required syntax, indicating that no command is to be executed when this key is pressed.

The next line shows the definition of PF3 as the SHOW STACK command. The use of the /IF_STATE qualifier makes the definition valid only when the command state is GREEN when PF3 is pressed, which means that PF1 must be pressed first. The use of the /TERMINATE qualifier causes the command to execute as soon as the PF3 key is pressed. SDA does not wait for RETURN to be pressed to terminate the command line.

EVALUATE

Computes and displays the value of the specified expression in both hexadecimal and decimal. If the expression is equal to the value of a symbol in the SDA symbol table, that symbol is displayed. A finite number of such symbols is displayed by default. 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.

Alternative evaluations of the expression are available with the use of the qualifiers defined for this command.

FORMAT

EVALUATE *expression*

command parameter

expression
The SDA expression to be evaluated. Section 6.2 defines SDA expressions.

command 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 displayed.

EXAMPLES

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

The preceding example shows how the EVALUATE command evaluates a numeric expression and displays the value of that expression in hexadecimal and decimal notation.

```
2   SDA> EVALUATE 1
    Hex = 00000001      Decimal = 1         ACP$V_SWAPGRP
                                            ACP$V_WRITECHK
                                            EVT$_EVENT
                                               .
                                               .
                                               .
```

The preceding example shows how the EVALUATE command evaluates a numeric expression and displays the value of that expression in hexadecimal and decimal notation. The preceding 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
```

The preceding 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
```

The preceding 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. See Section 6.2 for a detailed discussion of expressions.

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

The preceding 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
```

The preceding example shows the output of an EVALUATE/PSL command. SDA interprets the entered value 04080009 as though it were a program status longword and displays the resulting field values in that longword.

System Dump Analyzer
EVALUATE

```
SDA> EVALUATE/PTE ABCDFFEE

|31       28|27       24|23       20|19       16|15       12|11        8|7
|           |           |           |           |           |           |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-->
|1 |  0 1 0 1  |0 |--| 1 1 |--| 0|                        ODFFEE
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+-->
Vld  Prot= EW   M     Own=U     W                     Page Frame Number
          Page is Active and Valid
```

The preceding example shows the output of an EVALUATE/PTE command, which shows how SDA displays the page table entry and labels the fields. It also describes the page status.

EXAMINE

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

FORMAT

EXAMINE *[/qualifier[,...]] [parameter]*

command parameters

location

The location in memory to be examined. The default value of this parameter is initially 0, and subsequently is 4 plus the last address examined. Subsequent default addresses are affected by the /INSTRUCTION qualifier.

m:n

A range of locations to be examined, from m to n.

m;n

A range of locations to be examined, starting at m and continuing for n *bytes*.

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 may skip some bytes at the beginning of the range to ensure that SDA is properly synchronized with the start of each instruction. This synchronization may be overridden by specifying the /NOSKIP qualifier with the /INSTRUCTION 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, the default address parameter is initially 0. 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/INSTRUCTION command not to skip any bytes in the range when translating the contents of memory into VAX 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
Prints 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 EXAMINE command displays the contents of memory and registers. The following sections describe how to use this command.

Examining Locations

Use the location parameter to examine a specific location. A location can be represented by any valid SDA expression.

To examine a range of locations, designate starting and ending locations separated by a colon, for example, G40:G200; or specify a location and a length, in bytes, separated by a semicolon, for example, G400;16.

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

```
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
```

The contents of addresses 100 through 1FF are nonexistent, and are so labeled in the message. Addresses 200 through 220 do exist, and SDA displays their contents. Note that the ASCII representation of the contents of memory reads from left to right, but the hexadecimal representation reads from right to left. Thus, for example, the byte at 210 contains ..!. (unprintable ASCII characters are represented by periods), which is 04210218 (hexadecimal).

If a range of virtual locations contains only zeros, SDA prints the message:

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

SDA initially sets the current location to zero in the program region (P0) of either the process that was executing at the time the system failed (if you are examining a crash dump), or your process (if you are examining the running system). To examine memory locations of other processes, you must use the SET PROCESS command.

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.

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 message:

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

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

You can use the SHOW STACK command to interpret the contents of memory as symbolic addresses. See the description of the SHOW STACK command for details.

Examining Memory Regions

You can display an entire region of virtual memory by using one or more qualifiers with the EXAMINE command.

SDA displays six columns of information. The first four columns contain the contents of memory, in hexadecimal. Each column represents a longword.

The fifth column contains the ASCII value of each byte in the longwords of the first four columns. Where the ASCII character that corresponds to the value contained in a byte is not a printing character, SDA prints a period character (.) in column 5.

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 is the ASCII representation of the longwords. Thus, you read the dump display from right to left, and the ASCII display from left to right. See the following example for an illustration.

Examining the PSL

To examine the processor-status longword, use the /PSL qualifier with the EXAMINE command.

EXAMPLES

1
```
SDA> EXAMINE/SYSTEM
System Region Memory
--------------------
00040039 8FBC0010 00040038 8FBC0010  ....8.......9...      800000000
    .
    .
    .
```

The preceding 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. The ASCII value of 38, 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. The ASCII value of 39 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
```

The preceding example shows the display produced by the EXAMINE/PSL command. The address of the longword examined is 80001268.

EXIT

Performs two functions: it exits from an SDA display, and it exits from the utility.

FORMAT

EXIT

command parameters

None.

command qualifiers

None.

DESCRIPTION

If SDA is displaying information on a video display terminal such as a VT100, and if that display is more than one page of information, SDA displays the following message, called a screen overflow prompt, each time it reaches the bottom of a page:

```
Press RETURN for more.
SDA>
```

If you want to discontinue the current display at this point, type EXIT. (On printing terminals, SDA does not display a prompt at the bottom of each page.) If you want SDA to execute another command at this point, type that command. SDA discontinues the display as if you typed EXIT, and then executes the command you typed.

To stop SDA, type EXIT in response to the SDA> prompt.

FORMAT

The FORMAT command displays a formatted list of the contents of a block of memory. It attempts to

- Characterize a range of locations as a systemwide data block
- Assign a symbol to each item of data within the block
- Display all the data within the block

FORMAT

FORMAT *[/qualifier] location*

command parameter

location

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

command qualifier

/TYPE=block-type

This qualifier indicates the symbolic prefix that corresponds to the type of block structure you want to format. This qualifier accepts as parameters the prefix of any VAX/VMS control block. See the READ command for a list of the symbol-table files supplied with the VAX/VMS operating system.

DESCRIPTION

The FORMAT command causes SDA to examine the byte at location+10 (decimal) or location+A (hexadecimal), which contains the type of the data block in most systemwide data blocks. If this byte contains a valid block type, SDA checks the next byte, at location+11, for the secondary block type. If byte 10 does not contain a valid block type, no further action is taken.

When SDA has determined the type of block, it tries to find the symbols that correspond to that type of block.

If SDA cannot find the symbols associated with the block type you have indicated or that it has found in the the block you specified, it issues the message:

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

Not every data block contains its type byte at offset 10. If this byte is absent or contains information other than a block type, or the byte does not contain a valid block type, SDA produces the message:

```
Invalid block type in specified block
```

To format such a block, you must retype the FORMAT command, using the /TYPE= qualifier to designate a block type, or use the READ command to have SDA read the file that contains the definitions of the symbols; and then retype the FORMAT command.

The display produced by the FORMAT command shows, from left to right, the virtual address of each item within the block, its symbolic name, and its hexadecimal representation.

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_FIPL              08
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
   .
   .
   .
```

The FORMAT command displays the data structure that begins at 800B81F0, a UCB. SDA uses the symbols in SYSDEF.STB to provide the names displayed next to each address displayed. If the field has more than one symbolic name, all such names are displayed. Thus, the field that starts at 800B8204 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 appears to the right of the symbolic name of the field. Thus, the contents of UCB$L_FIRST are 8002CA00.

HELP

Displays information about the SDA utility, its operation, and the format of its commands. HELP has three command parameters. If you do not specify a parameter, HELP gives a brief description of SDA operations and displays SDA commands.

FORMAT

HELP *[parameter]*

command parameters

command-name
Specifies the command for which you need information.

EXPRESSION
Prints a description of SDA expressions.

OPERATION
Describes how to operate SDA at your terminal and by means of the site-specific startup procedure.

command qualifiers

None.

READ

Causes SDA to read the global symbols contained in the specified object module and to add those symbols to the SDA symbol table. SDA extracts no local symbols from the object module.

FORMAT

READ *[/qualifier[,...]] filespec*

command parameter

filespec
SYS$DISK:[default-dir]filename.stb

The name of the device, directory, and file that contains the object module from which you want to copy global symbols. The object module file can be the output of a compiler or assembler, the output generated by the linker qualifier /SYMBOL_TABLE, or one of the object module files provided by VAX/VMS. Those files are the following:

File	Contents
DCLDEF.STB	The symbols for the DCL command language interpreter.
IMGDEF.STB	Symbols for the image activator.
NETDEF.STB	Symbols that define DECnet data structures.
RMS.STB	Global symbols for VAX RMS.
RMSDEF.STB	Symbols that define RMS internal and user data structures. Also, contains the RMS$_xxx completion codes.
MP.STB	Symbols for multiprocessor code.
SCSDEF.STB	Symbols that define data structures for system communications services.
SYSDEF.STB	Symbols that define system data structures, including the I/O database.

command qualifier

/RELOCATE=expression

Add the value of **expression** to the value of each symbol in the symbol-table file to be read. This qualifier is useful for examining images that are position independent and are loaded at a base of zero.

DESCRIPTION

The READ command is useful in those cases where the symbols needed are defined in modules that are compiled and linked separately from the VAX/VMS executive.

EXAMPLES

1
```
SDA> READ SYS$SYSTEM:SYSDEF.STB
SDA>
```

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 use the FORMAT command, for example.

2
```
SDA> READ/RELOCATE=MP SYS$SYSTEM:MP.STB
SDA>
```

This READ command causes SDA to read the file that contains the symbols defined for the code that supports multiprocessors. The /RELOCATE qualifier causes the values of those symbols to be the sum of the value of the symbol in the file and the value of symbol MP, the address of the beginning of the multiprocessor code.

REPEAT

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

FORMAT

REPEAT

command parameters

None.

command 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.

EXAMPLE

```
SDA> FORMAT G10E0
8000010E0  UCB$L_FQFL          80002428
           UCB$L_FQBL
    .
    .
    .
SDA> KP0
SDA> FORMAT G10E0
8000010E0  UCB$L_FQFL          80002428
           UCB$L_FQBL
    .
    .
    .
```

The FORMAT command displays a UCB. The REPEAT command causes SDA to display the FORMAT command, then reexecute it, displaying the contents of the UCB once more.

SEARCH

Scans a range of memory locations for all occurrences of a longword value. SEARCH displays each location as each value is found.

FORMAT

SEARCH *range[=]expression*

command parameters

range

The range of memory locations that you want SDA to search. You can specify a range as a starting location and an ending location, separated by a colon (:), or as a location and a length, in bytes, separated by a semicolon (;).

expression

An expression. SDA evaluates this expression and searches the range of memory for that value. SDA only searches for values that are aligned on longword boundaries. For a definition of SDA expressions, see Section 6.2.

command qualifier

/LENGTH=length_specifier

The /LENGTH qualifier specifies the size of the expression value to be used for successful matching during searches of memory. The possible values of this qualifier are:

- LONGWORD — specifies that the expression for which to search is 4 bytes in length. This is the default value.
- WORD — specifies that the expression for which to search is 2 bytes in length.
- BYTE — specifies that the expression for which to search is 1 byte in length.

/STEPS=step_factor

The /STEPS qualifier controls the granularity of searching through the specified memory range. As each comparison of memory occurs, the value of this qualifier determines the next memory location to be searched. The possible step factors are as follows:

- QUADWORD — specifies a step factor of 8 bytes.
- LONGWORD — specifies a step factor of 4 bytes. This is the default value for the /STEPS qualifier.
- WORD — specifies a step factor of 2 bytes.
- BYTE — specifies a step factor of 1 byte.

EXAMPLES

1
```
SDA> SEARCH GB81FO;500 60068
Searching from 800B81FO to 800B86FO in LONGWORD steps for 00060068...
Match at 800B8210
SDA>
```

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

SET LOG

Sends output from SDA to both your terminal and to a log file.

FORMAT

SET LOG *file-spec*

command parameter

file-spec

The name of the file in which you want SDA to log your commands and their output. The default device is SYS$OUTPUT.

command qualifiers

None.

DESCRIPTION

The SET LOG command sends your commands and the output they produce to a log file. Your commands and the resulting displays are still displayed on your terminal. Note that any output redirected to another file by means of the SET OUTPUT command does not appear in the log file.

SET NOLOG

Stops SDA from logging your commands and their output.

FORMAT	SET NOLOG
command parameters	*None.*
command qualifiers	*None.*

SET OUTPUT

Redirects the SDA output to the file or device of your choice.

FORMAT

SET OUTPUT *file-spec*

command parameter

file-spec

The name of a device or the specification of a file to which you want SDA to send the output generated by your commands. The default file specification is SYS$DISK:[default-dir] SYSDUMP.LIS.

DESCRIPTION

When you use the SET OUTPUT command to send the SDA output to a file or device, SDA continues displaying the SDA commands that you type but sends the output generated by those commands to the file or device you specified.

If you finish directing SDA commands to an output file and wish to return to interactive display, issue the command SET OUTPUT TT.

If you use the SET OUTPUT command to send the SDA output to a file, SDA remembers all the commands you use to show information until you either use another SET OUTPUT command or exit from the utility. SDA then builds a table of contents that identifies the displays you selected and places the table of contents at the beginning of the output file.

EXAMPLE

```
SDA> SET OUTPUT DUMPDISK:[CURRENT]SDA.TXT
SDA> SHOW CRASH
SDA> EXIT
$ TYPE DUMPDISK:[CURRENT]SDA.TXT



VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 15:22:29.13   Page 1
    .
    .
    .
VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 15:22:29.13   Page 2
System crash information

Time of system crash: 06-JAN-1986 15:21:53.38

Version of system: VAX/VMS VERSION X4.4

VAXcluster node name: REDDOG
Process currently executing: CRAWDAD
Current image file: $254$DUSO:[SYS4.SYSCOMMON.][SYSEXE]SDA.EXE;1
Current IPL: 0  (decimal)

General registers:
        R0  = 00000000   R1  = 00000000   R2  = 00000000   R3  = 00000000
        R4  = 00000000   R5  = 00000000   R6  = 00000000   R7  = 00000000
        R8  = 00000000   R9  = 00000000   R10 = 00000000   R11 = 00000000
        AP  = 00000000   FP  = 00000000   SP  = 00000000   PC  = 00000000
        PSL = 00000000
```

```
Processor registers:          VAX 8600
        POBR    = 00000000      SBR     = 00000000      ASTLVL = 00000000
        POLR    = 00000000      SLR     = 00000000      SISR   = 00000000
        P1BR    = 00000000      PCBB    = 00000000      ICCS   = 00000000
        P1LR    = 00000000      SCBB    = 00000000      SID    = 0404F00B

        ICR     = 00000000      SBISTS = 00000000       SBIERR = 00000000
        TODR    = 00000000      SILOCMP= 00000000       TMOADDRS=00000000
        ACCS    = 00000000      MAINT  = 00000000

        ISP     = 00000000
        KSP     = 00000000
        ESP     = 00000000
        SSP     = 00000000
        USP     = 00000000

VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 15:22:29.13      Page 3
System crash information

SBI silo contents:

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

The preceding example shows the SET OUTPUT command as used to redirect output from the SHOW CRASH command to a file called SDA.TXT on disk DUMPDISK in directory [CURRENT]. The TYPE command shows the contents of the file that SDA generates.

SET PROCESS

The SET PROCESS command changes process context, making the specified process the SDA current process.

FORMAT

SET PROCESS *[/qualifier[,...]] [name]*

command parameter

name

The name of the process to become the SDA current process. The name is a string containing up to 15 uppercase alphabetic characters or numerals. The dollar sign ($) and underscore (_) characters can be included in the string. The name must be a quoted string if other than the foregoing characters are included.

command qualifiers

/INDEX=nn

The index into the system's list of software process control blocks (PCB). Alternatively, this argument can be the process identification (EPID or PID) longword, from which SDA extracts the correct index.

/SYSTEM

The system process control block. The system PCB and process header (PHD) are dummy structures that are located in system space and contain the system working set, global section table, global page table, and other systemwide data.

DESCRIPTION

When you issue an SDA command, for example 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.

This command allows you to examine the data structures associated with any process. The process specified by this command becomes the SDA current process until you either use another SET PROCESS command or exit from SDA.

When you invoke SDA to examine a system dump, the SDA current process is the process that was executing when the system failed. If you invoke SDA to examine the running system, the current process is your process.

SET PROCESS locates the information needed for the particular process but produces no output.

You must specify one of the three SET PROCESS parameters and qualifiers, or SDA generates a syntax error.

EXAMPLES

1

```
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
   .
   .
   .
```

2

```
SDA> SHOW SUMMARY
Current process summary
-----------------------
 Extended Indx Process name    Username    State Pri   PCB      PHD     Wkset
-- PID -- ---- --------------- ----------- ----- --- -------- -------- -----
 28C00080 0000 NULL                         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
   .
   .
   .
```

3

```
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 other processes that exist. The SET PROCESS command sets the current process to ERRFMT, as shown by the second SHOW PROCESS command.

SET RMS

The SET RMS command changes the options shown by the SHOW PROCESS/RMS command.

FORMAT

SET RMS=*option*

command parameter

option

The list of RMS data structures to be shown by the SHOW RMS command. The options are listed in the table that follows. You can suppress output for a given option by preceding that option with NO.

The list can consist of one or more options. If the list contains more than one option, the list must be in parentheses, and options must be separated by commas.

The optional parameter **ifi** is an internal file identification. The default ifi is all the files the current process has opened.

List-item	Meaning
ALL[:ifi]	All control blocks, the default
ASB	Asynchronous-save block
BDB	Buffer-descriptor block
BDBSUM	BDB summary page
BLB	Buffer-lock block
BLBSUM	Buffer-lock summary page
CCB	Channel-control block
FCB	File-control block
FWA	File work area
GBDSUM	GBD summary page
GBH	Global buffer's header
IDX	Index descriptor
IFAB[:ifi]	Internal FAB
IFB[:ifi]	Internal FAB
IRAB	Internal RAB
IRB	Internal RAB
RLB	Record-lock block
TRC	Global-buffer trace information
WCB	Window-control block
*	Current list of options displayed by the SHOW RMS command

command qualifiers

None.

DESCRIPTION

The SET RMS command determines the data structures to be displayed by the SHOW PROCESS/RMS command. The options you specify with this command are the types of data structures that will be displayed, and any options not specified in the command will not be displayed. The initial list of options is that specified by the ALL parameter.

To add or delete an option from the current list to be displayed, without having to specify the entire list, use the asterisk parameter (*) and one or more other options. The asterisk paramenter (*) represents the current list.

EXAMPLES

1

```
SDA> SET RMS=(WCB,CCB,BDB,ASB)
SDA> SHOW PROCESS/RMS


Process index: 003C   Name: BIRDSONG   Extended PID: 212000BC
------------------------------------------------------------

CCB Address: 7FFDAF40
------------

UCB:            80159960                WIND:           802A6080
STS:            00
AMOD:           02          Executive
IOC:            0000            0.      DIRP:           00000000

WCB Address: 802A6080
------------

WLFL:           802B5B90                SIZE:           0060          96.
WLBL:           802B5B90                TYPE:           12
ACCESS:         03          READ,WRITE
PID:            0001003C                ORGUCB:         80159960
ACON:           0501        NOWRITE,WRITEAC,NOREAD
NMAP:           0001            1.      FCB:            802B5B80
STVBN:          00000001        1.      RVT:            00000000

         VBN     RVN       Starting LBN                Count
         ---     ---       ------------                -----

           1.     0   0002A5C7       173511.        0021       33.

ASB Address: 7FF74600
------------

ARGCNT:         00              0.      BID:            0D            13.
FABRAB:         00000000                BLN:            2F            47.
ERR:            00000000                STKLEN:         008C         140.
SUC:            00000000                STKSIZ:         002C          44.

R6:             0000001E
R7:             00000003
R8:             000011BC
R10:            7FF73608
R11:            7FFE0270
```

```
Saved Stack:
------------
        SP =>  7FF74630  80046456
               7FF74634  0000164E      SGN$C_MAXPGFL+64E
               7FF74638  80041F7E
               7FF7463C  00000044
               7FF74640  7FF741A0
               7FF74644  00000001
               7FF74648  7FF74150
               7FF7464C  0000164E      SGN$C_MAXPGFL+64E
               7FF74650  80045D44
               7FF74654  00000001
               7FF74658  80045EDE

BDB Address: 7FF737A8
------------

FLINK:          7FF74150                BID:            0C          12.
BLINK:          7FF73648                BLN:            14          20.
FLGS:           00
USERS:          0000          0.        BLB_PTR:        00000000
CACHE_VAL:      00            0.        BUFF_ID:        0000          0.
SIZE:           2000       8192.        NUMB:           0000          0.
ADDR:           7FF74800                VBN:            00000000      0.
VBNSEQNO:       00000000                WAIT:           00000000
WK1:            00000000                CURBUFADR:      00000000
REL_VBN:        00            0.        PRE_CCTL:       00
VAL_VBNS:       00            0.        POST_CCTL:      00
JNLSEQ:         00000000  00000000  00000000  00000000
IOSB:           00000000
                00000000

BDB Address: 7FF74150
------------

FLINK:          7FF73648                BID:            0C          12.
BLINK:          7FF737A8                BLN:            14          20.
FLGS:           03          VAL,DRT
USERS:          0000          0.        BLB_PTR:        00000000
CACHE_VAL:      00            0.        BUFF_ID:        0000          0.
SIZE:           2000       8192.        NUMB:           0200        512.
ADDR:           7FF76800                VBN:            00000001      1.
VBNSEQNO:       00000000                WAIT:           00000000
WK1:            00000605                CURBUFADR:      7FF77200
REL_VBN:        05            5.        PRE_CCTL:       00
VAL_VBNS:       06            6.        POST_CCTL:      00
JNLSEQ:         00000000  00000000  00000000  00000000
IOSB:           00000605
                7FF77200
```

The preceding example shows a SET RMS command used to set the data structures that the SHOW PROCESS/RMS command displays. The subsequent SHOW PROCESS/RMS command displays the WCB, the CCB, the BDB, and the ASB.

```
2     SDA> SET RMS=(*,NOASB)
      SDA> SHOW PROCESS/RMS


Process index: 003C   Name: WILLING   Extended PID: 212000BC
------------------------------------------------------------

CCB Address: 7FFDAF40
------------

UCB:            80159960                WIND:           802A6080
STS:            00
AMOD:           02          Executive
IOC:            0000          0.        DIRP:           00000000
```

```
WCB Address: 802A6080
------------

WLFL:           802B5B90                SIZE:           0060          96.
WLBL:           802B5B90                TYPE:           12
ACCESS:         03          READ,WRITE
PID:            0001003C                ORGUCB:         80159960
ACON:           0501        NOWRITE,WRITEAC,NOREAD
NMAP:           0001            1.      FCB:            802B5B80
STVBN:          00000001        1.      RVT:            00000000
         VBN     RVN       Starting LBN                Count
         ---     ---       ------------                -----

           1.     0    0002A5C7       173511.        0021       33.

BDB Address: 7FF737A8
------------

FLINK:          7FF74150                BID:            0C            12.
BLINK:          7FF73648                BLN:            14            20.
FLGS:           00
USERS:          0000            0.      BLB_PTR:        00000000
CACHE_VAL:      00              0.      BUFF_ID:        0000           0.
SIZE:           2000         8192.      NUMB:           0000           0.
ADDR:           7FF74800                VBN:            00000000       0.
VBNSEQNO:       00000000                WAIT:           00000000
WK1:            00000000                CURBUFADR:      00000000
REL_VBN:        00              0.      PRE_CCTL:       00
VAL_VBNS:       00              0.      POST_CCTL:      00
JNLSEQ:         00000000  00000000  00000000  00000000
IOSB:           00000000
                00000000

BDB Address: 7FF74150
------------

FLINK:          7FF73648                BID:            0C            12.
BLINK:          7FF737A8                BLN:            14            20.
FLGS:           03          VAL,DRT
USERS:          0000            0.      BLB_PTR:        00000000
CACHE_VAL:      00              0.      BUFF_ID:        0000           0.
SIZE:           2000         8192.      NUMB:           0200         512.
ADDR:           7FF76800                VBN:            00000001       1.
VBNSEQNO:       00000000                WAIT:           00000000
WK1:            00000A09                CURBUFADR:      7FF77A00
REL_VBN:        09              9.      PRE_CCTL:       00
VAL_VBNS:       0A             10.      POST_CCTL:      00
JNLSEQ:         00000000  00000000  00000000  00000000
IOSB:           00000A09
                7FF77A00
```

In the preceding example, the SET RMS command sets the data structures that the SHOW PROCESS/RMS command shows to be all the data structures currently selected for display [indicated by the asterisk paramenter (*)] except the ASB (indicated by the NOASB parameter).

SHOW CLUSTER

Displays a view of the VAXcluster or the system communications services (SCS) cluster. You can display information for all of the nodes in a VAXcluster, a specific node in a VAXcluster, or information about the cluster as seen by the SCS.

FORMAT

SHOW CLUSTER

command parameters

None.

command qualifiers

/CSID=n

Displays cluster information for a specific VAXcluster member node. The value that you specify to obtain information for a specific node is that node's cluster system identification number (CSID).

You can find the CSID for a specific node in a VAXcluster by examining the first display on your output device after you issue a SHOW CLUSTER command.

If you want to obtain CSID information to indicate where a lock is mastered or held, use the SHOW LOCK command.

/SCS

Displays a view of the cluster as seen by the systems communications services (SCS).

DESCRIPTION

The SHOW CLUSTER command provides a series of displays to your default or designated output device.

The first display is a summary of the VAXcluster. This summary includes the number of votes required for a quorum, the number of votes currently available, the number of votes allocated to the quorum disk, and a status summary indicating whether or not a quorum is present. Additionally, the initial SHOW CLUSTER display lists the cluster system blocks (CSB) currently in operation; there is one CSB assigned to each node of the VAXcluster. For each CSB, the first SHOW CLUSTER display shows the node name, the associated CSB address, the CSID associated with the node, the number of votes (if any) provided by the node, its state, and its status. (For information about the state and status of nodes, see the description of the ADD command in the *VAX/VMS Show Cluster Utility Reference Manual*.)

The second display for the SHOW CLUSTER command describes the cluster block (CLUB). The information provided includes a list of flags that have been activated, a summary of quorum and vote information, and other data that applies to the VAXcluster from the perspective of the node for which the SDA is being run.

The next display provides information concerning the cluster failover control block (CLUFCB) and the cluster quorum disk control block (CLUDCB).

Subsequent displays provide information for the individual cluster system blocks (that is, for the individual nodes) representing members of the VAXcluster; each CSB is a separate display. For each CSB, its state and flags are shown, as well as other information specific to each node. (Information about the flags for nodes of a VAXcluster is provided in the *VAX/VMS Show Cluster Utility Reference Manual*.)

You can obtain information about a specific node of the cluster with the /CSID=n qualifier, using the CSID value as shown in the first display of the SHOW CLUSTER command. (You can also obtain this information by using the SHOW LOCKS command.)

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

The initial display for the SHOW CLUSTER /SCS command provides an overview of processes that are listening for incoming SCS connect requests. For each of these processes, this first display shows its entry address, connection ID, process name, and explanatory text, if any.

The second display with the /SCS qualifier is a summary of SCS systems. These systems can include cluster members (as shown in the SHOW CLUSTER command), HSCs, UDAs, and other such devices. For each of these SCS systems, the system block (SB) address, node name, system type, system ID, and the number of connection paths are shown.

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 Summary ---
        Quorum   Votes   Quorum Disk Votes   Status Summary
        ------   -----   -----------------   --------------
          2        3            1            quorum
                      --- 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-1986
Found Node SYSID     0000000008A0                              18:52:32
Founding Time          3-DEC-1985    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-1985    TQE address     00000000
Ref. time  18-DEC-1985                    08:52:20    SB address      8041B6E0
             08:53:58    Lock mgr dir wgt       1    Current CDRP    00000000
```

The preceding 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 the preceding example.)

```
2    SDA> SHOW CLUSTER /CSID=000100C8


        --- 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-1985    TQE address     00000000
Ref. time  18-DEC-1985                    08:52:20    SB address      8041B6E0
             08:53:58    Lock mgr dir wgt       1    Current CDRP    00000000
```

The preceding example shows the use of the /CSID qualifier to obtain information about a specific node (in this instance, node SOLLY). The information displayed is identical to that shown for the specified node in the SHOW CLUSTER command.

```
SDA> SHOW CLUSTER /SCS

                --- 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.         X25C
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
Reseting 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.         V4.1
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
Reseting port                 02      Port dev. name               PAB0
Handshake retry cnt.           1      SCS MSGBUF address        8036F0B0
Msg. buf. wait queue         empty    PDT address               801CEA20
```

The preceding example shows a subset of a typical output for the SHOW CLUSTER /SCS command. In this system, there are three VAX/VMS 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 all active connections between systems communication services (SCS) processes. This information is retrieved from connection descriptor tables (CDTs). You can also display information for a specific CDT to obtain information about an individual connection.

FORMAT SHOW CONNECTIONS

command parameters

None.

command qualifiers

/ADDRESS=n

Displays information for a specific CDT. The addresses for individual CDTs are listed in the CDT summary page, which is the first display provided for the SHOW CONNECTIONS command.

DESCRIPTION

The SHOW CONNECTIONS command provides a series of displays to your default or designated output device.

The first display is a summary of the connection descriptor tables (CDTs). For each CDT, the display lists its address, the local process with which the CDT is associated, the connection ID, its current state, and the remote node to which it is currently connected (if any). This display also shows the number of CDTs that are currently free and available to the system.

CDT addresses, in addition to being available from the summary page shown by the SHOW CONNECTIONS command, are also stored in many individual data structures related to SCS connections. These data structures include CDRPs and UCBs for class drivers that use SCS and CSBs for the connection manager.

Next, there is a display of detailed information for each CDT that is listed on the first page summary. This information includes the current state, the associated local process, the associated remote node and process (if any), and detailed connection information. (See Example 1 for a list of the information available.)

You can obtain information for the individual connection between two SCS processes by using the /ADDRESS=n qualifier, obtaining the appropriate address either from the CDT summary table in the first display of the SHOW CONNECTIONS command or from an appropriate data structure.

EXAMPLES

1

```
SDA> SHOW CONNECTIONS
                        --- 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 CDT's:  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
```

The preceding example shows the first display, the CDT summary page, and the first page of the detailed displays for each CDT. A similar description of each CDT is provided when you issue the SHOW CONNECTIONS command.

2

```
SDA> SHOW CONNECTIONS /ADDRESS=801C27B0
             --- 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
Send Credit            5   Messages Sent    267   PDT address     801CEA20
Min. Rec. Credit       0   Messages Rcvd.   289   Error Notify    80227950
Pend Rec. Credit       1   Send Data Init.    0   Receive Buffer  8039AF80
Initial Rec. Credit    5   Req Data Init.     0   Connect Data    80367C0C
Rem. Sta.   000000000002   Bytes Sent         0   Aux. Structure  80367B90
Rej/Disconn Reason     0   Bytes rcvd         0
Queued for BDT         0   Total bytes map    0
Queued Send Credit     0
```

The preceding example shows the use of the /ADDRESS qualifier to obtain information about a specific connection. The address that you use to specify the table is obtained either from the CDT summary page or an appropriate data structure.

SHOW CRASH

Displays information concerning the operating system and the currently executing process. The display shows the following:

- Operating system and process information
- General and special register contents
- Processor and hardware maintenance register contents

FORMAT

SHOW CRASH

command parameters

None.

command qualifiers

None.

DESCRIPTION

The SHOW CRASH command displays information in three sections. The contents of each display is described here:

Operating System and Process Information

The first section of the SHOW CRASH display lists the following:

- Date and time of the crash
- Name and version number of the operating system
- Reason for the bugcheck
- Name of the currently executing process
- Specification of the file that contains the image executing in the process context (left blank if no image is executing)
- Interrupt priority level (in decimal) of the processor

Contents of General and Special Registers

The second section of the SHOW CRASH display lists the contents of the general registers and the special registers as follows:

- R0 through R11
- Argument pointer (AP)
- Frame pointer (FP)
- Stack pointer (SP)
- Program counter (PC)
- Processor status longword (PSL)

Contents of Process and Hardware-Maintenance Registers

The third section of the SHOW CRASH display lists the contents of three sets of registers. The first set includes registers that store the vital statistics of the currently executing process, as well as registers that contain information used by the operating system. The second set of registers are pointers to the five stacks, those being the interrupt stack and the stack for each processor access mode. The third set of registers are used in hardware maintenance.

Each type of VAX processor supports a different set of hardware (processor) registers, but all have the same process and system registers, and the same stack pointers. In any case, the processor type is displayed.

The process and system registers are as follows:

- Program region base register (P0BR)
- Program region length register (P0LR)
- Control region base register (P1BR)
- Control region length register (P1LR)
- System region base register (SBR)
- System region length register (SLR)
- Process control block base register (PCBB)
- System control block base register (SCBB)
- Asynchronous system trap level (ASTLVL)
- Software interrupt summary register (SISR)
- Internal clock control/status register (ICCS)
- System identification register (SID)

The stack pointers are as follows:

- Interrupt stack pointer (ISP)
- Kernel-mode stack pointer (KSP)
- Executive-mode stack pointer (ESP)
- Supervisor-mode stack pointer (SSP)
- User-mode stack pointer (USP)

EXAMPLE

```
SDA> SHOW CRASH
System crash information
------------------------
VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 15:22:29.13   Page 1
    .
    .
    .
VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 15:22:29.13   Page 2
System crash information
Time of system crash: 06-JAN-1986 15:21:53.38
Version of system: VAX/VMS VERSION X4.4
VAXcluster node name: REDDOG
```

```
Process currently executing: CRAWDAD
Current image file: $254$DUSO:[SYS4.SYSCOMMON.][SYSEXE]SDA.EXE;1
Current IPL: 0  (decimal)

General registers:
        R0  = 00000000   R1  = 00000000   R2  = 00000000   R3  = 00000000
        R4  = 00000000   R5  = 00000000   R6  = 00000000   R7  = 00000000
        R8  = 00000000   R9  = 00000000   R10 = 00000000   R11 = 00000000
        AP  = 00000000   FP  = 00000000   SP  = 00000000   PC  = 00000000
        PSL = 00000000

Processor registers:         VAX 8600
        POBR   = 00000000      SBR    = 00000000      ASTLVL = 00000000
        POLR   = 00000000      SLR    = 00000000      SISR   = 00000000
        P1BR   = 00000000      PCBB   = 00000000      ICCS   = 00000000
        P1LR   = 00000000      SCBB   = 00000000      SID    = 0404F00B

        ICR    = 00000000      SBISTS = 00000000      SBIERR = 00000000
        TODR   = 00000000      SILOCMP= 00000000      TMOADDRS=00000000
        ACCS   = 00000000      MAINT  = 00000000

        ISP    = 00000000
        KSP    = 00000000
        ESP    = 00000000
        SSP    = 00000000
        USP    = 00000000

VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 15:22:29.13      Page 3
System crash information

SBI silo contents:
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
                00000000
```

The SHOW CRASH command displays the preceding information for a running system.

SHOW DEVICE

Displays a list of all data structures associated with a device.

FORMAT

SHOW DEVICE *[device-name]*

command parameter

device-name

The name of the device for which you want information. If you do not include this parameter, this command displays information on all devices in the system.

command qualifier

/ADDRESS=n

The address of the UCB of the device of interest. Using this qualifier with the SHOW DEVICE command is equivalent to specifying the name of the device with the command.

DESCRIPTION

The SHOW DEVICE command displays three data structures listed next:

- The device data block
- The controller data structure
- The unit data structures

If you provide the name of a device as a parameter to this command, the information is displayed for that device. In a cluster environment, the information is displayed for each device in the cluster with that name. If you provide no parameter, the information is displayed for every device configured in the system.

If you omit part of a device name, all devices that have a part of their device name matching that which you specified are displayed by the SHOW DEVICE command.

For a detailed explanation of I/O data structures displayed by SDA, consult the manual entitled *Writing a Device Driver for VAX/VMS*.

EXAMPLES

1

```
SDA> SHOW DEVICE VTA100
I/O data structures
-------------------
VTA100 ==> LTA83                        VT200_Series      UCB address:  802292B0
Device status:   00010010 online,deleteucb
Characteristics: 0C040007 rec,ccl,trm,avl,idv,odv
                 00000200 nnml
```

System Dump Analyzer
SHOW DEVICE

```
Owner UIC [000001,000004]   Operation count        491   ORB address     80229360
      PID         00060079  Error count              0   DDB address     804A3EC0
Class/Type           42/6E  Reference count          9   DDT address     80288743
Def. buf. size          80  BOFF                  0273   CRB address     804A6AA0
DEVDEPEND         180013A0  Byte count            0200   AMB address     80240910
DEVDEPND2         7BF2100C  SVAPTE             8025BAC0  I/O wait queue     empty
FIPL/DIPL            08/08  DEVSTS                0000

       *** I/O request queue is empty ***
SDA>
```

2 `SDA> SHOW DEVICE DUSO`

```
VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 13:38:43.21     Page 1

ACTI$DUSO                               RA81              UCB address:  80000FF8
Device status:    00001810 online,valid,unload
Characteristics: 1C4D4008 dir,fod,shr,avl,mnt,elg,idv,odv,rnd
                 00000221 clu,mscp,nnm

Owner UIC [000001,000001]   Operation count      95308   ORB address     80000FA0
      PID         00000000  Error count              0   DDB address     80000F5C
Alloc. lock ID    00030002  Reference count        110   DDT address     801D5548
Alloc. class           254  Online count             0   VCB address     80195C30
Class/Type           01/15  Retry cnt/max          8/8   CRB address     803943E0
Def. buf. size         512  BOFF                  0000   PDT address     802C7020
DEVDEPEND         04E00E33  Byte count            0200   CDDB address    80178AD0
DEVDEPND2         00000000  SVAPTE             81B1BFA0  I/O wait queue     empty
FIPL/DIPL            08/08  DEVSTS                0004
                            RWAITCNT              0000

          --- Primary Class Driver Data Block (CDDB) 80178AD0 ---

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      80337830   DDB address     80000F5C
System ID        0000FFF2   Restart Queue      empty   CRB address     803943E0
                    0000   DAP Count              3   CDDB link       8019C020
Contrl. ID       0000FFF2   Contr. timeout        75   PDT address     802C7020
                 01010000   Reinit Count           4   Original UCB    00000000
Response ID      E7BA0020   Wait UCB Count         0   UCB chain       8017B0B0
MSCP Cmd status 00001879

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

STATE CDRP/IRP    PID    MODE CHAN  FUNC    WCB      EFN    AST      IOSB     STATUS
 C    80334EC0  00060028  K  0000  000C  80395F40    6   80CA9888  0040FE10  0016
        readpblk func,pagio,virtual
 C    8032A4F0  0005005E  E  0000  000C  80398E80    6   80B37068  0040FE70  0016
        readpblk func,pagio,virtual
 C    80323740  00050047  E  0000  000C  803A3920    6   80A59410  0040FE50  0016
        readpblk func,pagio,virtual

                --- Volume Control Block (VCB) 80195C30 ---

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

Mount count             1   Rel. volume            0   AQB address     803A8480
Transactions           93   Max. files        222768   RVT address     80000FF8
Free blocks        199020   Rsvd. files            9   FCB queue       803081C0
Window size             7   Cluster size           1   Cache blk.      80195D20
Vol. lock ID     00010004   Def. extend sz.        5   Shadow mem. FL  803A88A0
Block. lock ID   02F20206   Record size            0   Shadow mem. BL  803A89C0
Shadow lock ID   00010005
```

```
VAX/VMS 4.4 -- System Dump Analysis          06-JAN-1986 13:38:43.21     Page 2
I/O data structures

                --- Shadow set $254$DUSO member summary ---

Volume: CERIUM
Physical unit     Primary path      Secondary path    Member status
-------------     ------------      --------------    -------------
$254$DUA200       ACTI              ANTIM             Shadow set member
$254$DUA201       ACTI              ANTIM             Copy in progress

                 --- ACP Queue Block (AQB) 803A8480 ---

ACP requests are serviced by the Extended QIO Processor (XQP)

Status: 14 defsys,xqioproc
Mount count           26    ACP type          f11v2    Request queue      empty
                            ACP class             0

                   *** ACP request queue is empty ***

ACTI$DUA200 (ANTIM$DUA200)              RA81              UCB address:  8017C3F0

Device status:   00000010 online
Characteristics: 1C4D4108 dir,rct,fod,shr,avl,mnt,elg,idv,odv,rnd
                 00000271 clu,2p,mscp,ssm,nnm

Owner UIC [000000,000000]   Operation count          6   ORB address    8017C4F6
     PID         00000000   Error count              1   DDB address    80397620
Alloc. lock ID   01A101F1   Reference count          1   DDT address    801D5548
Alloc. class          254   Online count             0   VCB address    803A88A0
Class/Type          01/15   BOFF                  0000   CRB address    803943E0
Def. buf. size        512   Byte count            0000   PDT address    802C7020
DEVDEPEND        04E00E33   SVAPTE            00000000   CDDB address   80178AD0
DEVDEPND2        00000000   DEVSTS                0004   2P_CDDB addr.  8019DD80
FIPL/DIPL           08/08   RWAITCNT              0000   2P_DDB address 80397440
                                                         I/O wait queue    empty

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

              --- Volume Control Block (VCB) 803A88A0 ---

Volume: CERIUM      (Member of shadow set $254$DUSO)
Status: 00
Copy sequence number: 0000  Copy type: 0 nocpy

Transactions          1     UCB address   8017C3F0    Virtual unit UCB  80000FF8
Relative volume       0     Work area     00000000    Virtual unit VCB  80195C30
AQB address     803A8480                  00000000    Shadow member FL  803A89C0
RVT address     80000FF8                              Shadow member BL  80195CC8

VAX/VMS 4.4 -- System Dump Analysis          06-JAN-1986 13:38:43.21     Page 3
I/O data structures

ACTI$DUA201 (ANTIM$DUA201)              RA81              UCB address:  8017C550

Device status:   00000010 online
Characteristics: 1C4D4108 dir,rct,fod,shr,avl,mnt,elg,idv,odv,rnd
                 00000271 clu,2p,mscp,ssm,nnm

Owner UIC [000000,000000]   Operation count          9   ORB address    8017C656
     PID         00000000   Error count              0   DDB address    80397620
Alloc. lock ID   00010008   Reference count          1   DDT address    801D5548
Alloc. class          254   Online count             0   VCB address    803A89C0
Class/Type          01/15   BOFF                  0000   CRB address    803943E0
Def. buf. size        512   Byte count            0200   PDT address    802C7020
DEVDEPEND        04E00E33   SVAPTE            81B19AAC   CDDB address   80178AD0
DEVDEPND2        00000000   DEVSTS                0004   2P_CDDB addr.  8019DD80
FIPL/DIPL           08/08   RWAITCNT              0000   2P_DDB address 80397440
                                                         I/O wait queue    empty

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

              --- Volume Control Block (VCB) 803A89C0 ---

Volume: CERIUM      (Member of shadow set $254$DUSO)
Status: 08 rebldng
Copy sequence number: 000A  Copy type: 1 copy

Transactions          1     UCB address   8017C550    Virtual unit UCB  80000FF8
Relative volume       0     Work area     00000000    Virtual unit VCB  80195C30
AQB address     803A8480                  00000000    Shadow member FL  80195CC8
RVT address     80000FF8                              Shadow member BL  803A88A0
```

The preceding example shows the display produced by the command SHOW DEVICE when a 4-character device name mnemonic (DUS0) is provided.

```
3    SDA> SHOW DEVICE DJ


I/O data structures
-------------------
                                DDB list
                                --------
        Address     Controller      ACP        Driver       DPT    DPT size
        -------     ----------      ---        ------       ---    --------
       808CC5C0     ANTI$DJA       F11XQP     DUDRIVER   8055F000   60F8
       808CC7A0     ANTI$DJS       F11XQP     DUDRIVER   8055F000   60F8
       80900640     CERIU$DJA      F11XQP     DUDRIVER   8055F000   60F8
       80903820     NOBI$DJA       F11XQP     DUDRIVER   8055F000   60F8
       80904AE0     SELE$DJA       F11XQP     DUDRIVER   8055F000   60F8
I/O data structures
-------------------
Controller: ANTI$DJA
--------------------
              --- ANTI System Block (SB) 808C9080 ---
System ID              00000000FFF2    Local software type          HSC
Max message size             66        Local software vers.         X5J6
Max datagram size            62        Local software incarn.    CE9DD540
Local hardware type        HS50                                  008E6F20
Local hardware vers. 02270222031A      SCS poller timeout           0007
                     022702270227      SCS poller enable mask         01

I/O data structures
-------------------
                     --- Path Block (PB) 808EDCE0 ---
                Status:  0008
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
Reseting port                    0B    Port dev. name               PAA0
Handshake retry cnt.              1    SCS MSGBUF address       80739670
Msg. buf. wait queue          empty    PDT address              806FA820
                  --- Device Data Block (DDB) 808CC5C0 ---
Driver name           DUDRIVER   Alloc. class        254   DDT address     8055F088
ACP ident                F11     SB address      808C9080   CONLINK addr.   808CC7A0
ACP class               PACK     UCB address     804D13E0

                                DDB list
                                --------
        Address     Controller      ACP        Driver       DPT    DPT size
        -------     ----------      ---        ------       ---    --------
       808CC5C0     ANTI$DJA       F11XQP     DUDRIVER   8055F000   60F8
       80900640     CERIU$DJA      F11XQP     DUDRIVER   8055F000   60F8
       80903820     NOBI$DJA       F11XQP     DUDRIVER   8055F000   60F8
       80904AE0     SELE$DJA       F11XQP     DUDRIVER   8055F000   60F8
```

```
I/O data structures
-------------------
Controller: ANTI$DJA
--------------------

                --- ANTI System Block (SB) 808C9080 ---

System ID             00000000FFF2    Local software type          HSC
Max message size                66    Local software vers.         X5J6
Max datagram size               62    Local software incarn.   CE9DD540
Local hardware type           HS50                             008E6F20
Local hardware vers. 02270222031A    SCS poller timeout           0000
                     022702270227    SCS poller enable mask         01
    .
    .
    .
```

The preceding example shows the display produced by allowing SHOW DEVICE to match all devices that use DJ as part of their name.

SHOW HEADER

Displays the header of the dump file.

FORMAT

SHOW HEADER

command parameters

None.

command qualifiers

None.

DESCRIPTION

The display produced by the SHOW HEADER command contains information taken from the header of the dump file.

EXAMPLE

```
SDA> SHOW HEADER
Dump file header
----------------
00000000 00000000 00000000 00000000 00000000 00000000 00000001 00000000  ................................  000006B0
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  000006D0
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  000006F0
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  00000710
    .
    .
    .
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  00000C70
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................  00000C90
```

The preceding example shows the display produced by the SHOW HEADER command.

SHOW LOCK

Displays a list of all locks in the system.

FORMAT

SHOW LOCK *lockid*

command parameters

lockid
The number of a specific lock.

command qualifier

/ALL
Lists all locks that exist in the system.

DESCRIPTION

The display produced by the SHOW LOCK command contains information on each lock in the system. The display is formatted in the same way as the display produced by the command SHOW RESOURCE/LOCKID.

EXAMPLE

```
SDA> SHOW LOCK
Lock database
-------------

Lock id:  00010001   PID:    00000000   Flags:   NOQUEUE SYNCSTS SYSTEM
Par. id:  00000000   Granted at    EX            CVTSYS
Sublocks:        4
LKB:      801F4F00
Resource:      5F535953 24535953    SYS$SYS_  Status:  NOQUOTA
 Length   16  00000000 00A94449    ID......
 Exec. mode   00000000 00000000    ........
 System       00000000 00000000    ........

Lock id:  00020002   PID:    00000000   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  00000000   Granted at    CR            NOQUOTA CVTSYS
Sublocks:        0
LKB:      80201680
Resource:      4C45445F 24535953    SYS$_DEL  Status:  NOQUOTA
 Length   17  30414244 24494850    PHI$DBA0
 Kernel mode  00000000 0000003A    :.......
 System       00000000 00000000    ........
   .
   .
   .
```

The preceding example shows the display produced by the command SHOW LOCK.

SHOW PAGE_TABLE

Displays a list of system page table entries that map virtual pages to physical pages. You can display a range of page table entries or the entire system page table.

The SHOW PAGE_TABLE command displays information in 132 columns, rather than 80 columns, and so is best suited for use at printing terminals, at video terminals that can display 132 columns, or as input for a file that can be printed later on a line printer.

FORMAT

SHOW PAGE_TABLE *[/qualifier[,...]] [range]*

command parameter

range

The range of virtual addresses for which SDA is to display page table entries. You can specify a range as a beginning address and an ending address, separated by a colon (:); or you can specify the range as an address and the number of bytes following that address, separated by a semicolon (;).

command 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 qualifier.

DESCRIPTION

The SHOW PAGE_TABLE command displays the contents of the system page table and the global page table. You can display a range of page table entries or the entire system page table.

This command displays information in 15 columns that form two groups. The left group contains information on virtual pages. The right group contains information on physical pages.

The left group contains information obtained from the system page table. Each line of this display lists characteristics of a virtual page as well as information needed for address translation. The headings of the columns listed in the display produced by the SHOW PAGE_TABLE command are explained in the following text.

Value	Meaning
ADDRESS	The system virtual address that marks the base of a virtual page.
SVAPTE	The system virtual address of the page table entry that maps this virtual page.
PTE	The contents of the page table entry, a longword that describes a system virtual page.
Type	Type of virtual page. There are eight types.

Type	Meaning
VALID	A valid page (in main memory).
TRANS	A transitional page (between main memory and page lists).
DZERO	A demand-allocated, zero-filled page.
PGFIL	A page within a paging file.
STX	A section table's index page.
GPTX	An index page for a global page table.
IOPAG	A page in the I/O address space.
NXMEM	A 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, a code, derived from bits in the PTE, that designates the type of access (read and/or write) granted to processor access modes (kernel, executive, supervisor, or user).
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 codes are listed following.

Code	Meaning
M	The page has been modified.
L	The page is locked into a working set.
K	The owner can access the page in kernel mode.
E	The owner can access the page in executive mode.
S	The owner can access the page in supervisor mode.
U	The owner can access the page in user mode.

If the virtual page has been mapped to a physical page, the right-hand section of the display includes information from the page frame number (PFN) database. Otherwise, the section is left blank. SDA organizes the 18 bytes of PFN data into 9 categories:

<table>
<tr><th>Category</th><th>Meaning</th></tr>
<tr><td>PAGTYP</td><td>Type of physical page, one of six types.
<table>
<tr><th>Page Type</th><th>Meaning</th></tr>
<tr><td>PROCESS</td><td>The page is part of process space.</td></tr>
<tr><td>SYSTEM</td><td>The page is part of system space.</td></tr>
<tr><td>GLOBAL</td><td>The page is part of a global section.</td></tr>
<tr><td>PPGTBL</td><td>The page is part of a process's page table.</td></tr>
<tr><td>GPGTBL</td><td>The page is part of a global-page table.</td></tr>
<tr><td>GBLWRT</td><td>The page is part of a global, writeable section.</td></tr>
</table></td></tr>
<tr><td>LOC</td><td>The location of the page within the system, one of eight types.
<table>
<tr><th>Location</th><th>Meaning</th></tr>
<tr><td>ACTIVE</td><td>The page is in a working set.</td></tr>
<tr><td>MDFYLST</td><td>The page is in the modified-page list.</td></tr>
<tr><td>FREELST</td><td>The page is in the free-page list.</td></tr>
<tr><td>BADLST</td><td>The page is in the bad-page list.</td></tr>
<tr><td>RELPEND</td><td>Release of the page is pending.</td></tr>
<tr><td>RDERROR</td><td>The page has had an error during an attempted read operation.</td></tr>
<tr><td>PAGEOUT</td><td>The page is being written into a paging file.</td></tr>
<tr><td>PAGEIN</td><td>The page is being brought into memory from a paging file.</td></tr>
</table></td></tr>
<tr><td>STATE</td><td>The byte that describes the state of the physical page.</td></tr>
<tr><td>TYPE</td><td>The 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.</td></tr>
<tr><td>REFCOUNT</td><td>A 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.</td></tr>
<tr><td>BAK</td><td>The address of the backing store; location on a disk device to which pages can be written.</td></tr>
<tr><td>SVAPTE</td><td>The virtual address associated with this page frame. The two SVAPTEs indicate a valid link between physical and virtual address space.</td></tr>
<tr><td>FLINK</td><td>The 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.</td></tr>
<tr><td>BLINK</td><td>The backward link within PFN database; also acts as an index into the working set list.</td></tr>
</table>

SDA indicates pages are inaccessible by displaying the following message, where n represents the number of inaccessible pages.

```
(-------- n NULL PAGES)
```

EXAMPLE

```
SDA> SHOW PAGE_TABLE
System page table
-----------------

        ADDRESS       SVAPTE     PTE      TYPE  PROT  BITS PAGTYP    LOC STATE TYPE  REFCNT   BAK       SVAPTE  FLINK  BLINK
        80000000     80B91C00 F8001ADD     VALID UR         K
        80000200     80B91C04 F8001ADE     VALID UR         K
        80000400     80B91C08 F8001ADF     VALID UR         K
        80000600     80B91C0C F8001AE0     VALID UR         K
        80000800     80B91C10 F8001AE1     VALID UR         K
        80000A00     80B91C14 EC001AE2     VALID UREW M     K
        80000C00     80B91C18 EC001AE3     VALID UREW M     K
        80000E00     80B91C1C F4001AE4     VALID URKW M     K
        80001000     80B91C20 F4001AE5     VALID URKW M     K
        80001200     80B91C24 F4001AE6     VALID URKW M     K
        80001400     80B91C28 F4001AE7     VALID URKW M     K
        80001600     80B91C2C F4001AE8     VALID URKW M     K
        80001800     80B91C30 F0001AE9     VALID URKW       K
```

The preceding example shows the output produced by the SHOW PAGE_TABLE command.

SHOW PFN_DATA

Displays information that is contained in the page lists and in the PFN database. This information is used in translating physical page addresses to virtual page addresses.

FORMAT

SHOW PFN_DATA *[number]*

command parameter

number

The number of the physical page for which you want to display information.

command qualifiers

/ALL

Displays all of the previous information. This is the default for the command.

/BAD

Displays the bad-page list.

/FREE

Displays the free-page list.

/MODIFIED

Displays the modified-page list.

/SYSTEM

Displays the PFN database. The information is ordered by page frame number, starting at PFN zero.

DESCRIPTION

The SHOW PFN_DATA command causes SDA to display information regarding the specified PFN. The display consists of the information listed in the example following.

```
SDA> SHOW PFN_DATA 1000
PFN   PTE ADDRESS   BAK     REFCNT FLINK BLINK   TYPE        STATE
----  ----------- --------  ------ ----- -----  ----------  ----------
1000    8034AA50  0040FFB8      1   0000 0236  00 PROCESS  07 ACTIVE
```

The items of information are the following:

Item	Contents
PFN	The page-frame number, the number of this physical page
PTE ADDRESS	The system virtual address of the page table entry that describes the virtual paged mapped into this physical page
BAK	The place to find information on this page when all links to this PTE are broken, either an index into a process section table or the number of a virtual block in the paging file
REFCNT	The number of references being made to this page
FLINK	The address of the next page in the list in which this virtual page currently resides
BLINK	The address of the previous page in the list in which this virtual page currently resides
TYPE	The type of virtual page, one of the following:

Code	Meaning
0	Process page
1	System page
2	Global, read-only page
3	Global, read/write page
4	Process page-table page
5	Global page-table page

Item	Contents
STATE	The state of the virtual page, one of the following:

Code	Meaning
0	The page is on the free list.
1	The page is on the modified list.
2	The page is on the bad-page list.
3	Release of the page to the free-page or modified list is pending.
4	An error occurred as the page was being read from the disk.
5	The modified-page writer is currently writing the page to the disk.
6	The page-fault handler is currently reading the page from the disk.
7	The 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
```

System Dump Analyzer

SHOW PFN_DATA

```
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
   .
   .
   .
```

The SHOW PFN_DATA command displays the information shown previously 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 the contents of the look-aside (SRP, IRP, and LRP) pools, the nonpaged dynamic storage pool, and the paged dynamic storage pool. You can display part or all of each pool.

FORMAT

SHOW POOL *[range]*

command parameter

range

The range of virtual addresses that you want to display. You can specify a range as two addresses separated by a colon (:), or as an address and a length, in bytes, separated by a semicolon (;).

command qualifiers

/ALL

Displays the entire contents of memory. This is the default.

/FREE

Displays the look-aside, paged, and nonpaged pools and shows the blocks that are currently available to the system.

/HEADER

Displays only the first 16 longwords of each block within the pool.

/IRP

Displays the pool of I/O request packets. Displays all blocks currently allocated (in use) within this pool.

/LRP

Displays the pool of long I/O request packets. Formats all blocks currently allocated (in use) within this pool.

/NONPAGED

Displays the nonpaged dynamic storage pool currently in use by the system.

/PAGED

Displays the paged dynamic storage pool currently in use by the system.

/SRP

Displays the pool of short I/O request packets. Formats all blocks currently allocated (in use) within this pool.

/SUMMARY

Displays a summary of the pools or portions of pool specified by the qualifiers shown previously. This qualifier shows the different types of blocks present, the total number of each, the decimal number of bytes in each block, and the number of bytes used in each pool.

/TYPE=block-type

Displays the blocks within pool that are of the specified type.

DESCRIPTION

The information contained in each of the three pools is shown in the same format. The contents of the display, from left to right, are listed as follows:

Column	Contents
Block type	The type of information contained in the block. SDA tries to interpret the block type; if it is unable to do so, SDA displays the block type as UNKNOWN.
Starting address	The virtual address that marks the start of the block.
Block size	The number (decimal) of bytes of memory allocated to the block. The block size is fixed in the SRP, IRP, and LRP pools, and is variable in the paged and nonpaged pools.
Contents	The contents of the block, arranged in four columns of longwords. The contents of the longwords are represented in hexadecimal. On each line of the display, the longwords are arranged from right to left, in the order of their addresses.
Contents	The contents of the block, arranged in one column of ASCII characters. The column is 16 characters wide. Each character represents the ASCII value of the bytes within the longwords on that line. If the ASCII value of a byte is not a printing character, SDA prints a period character (.) instead. The characters are arranged from left to right, in order of their addresses.

EXAMPLE

```
SDA> SHOW POOL
IRP      8013B240   160
                    07010000 000A0054 00000000 000A0054 T.......T.......
```

The preceding example shows two lines from a display produced by the SHOW POOL command. This block is an IRP; its starting address is 8013B240; and its length is 160 bytes. Note that the first byte of the longword at 8013B240, the rightmost byte in the hexadecimal display, contains 54, the ASCII value of the character uppercase T. This value is represented as the leftmost character in the ASCII portion of the display.

Similarly, the byte at location 8013B48, the ninth byte from the right in the hexadecimal display, also contains 54. It is represented by the T in the ninth place from the left in the ASCII display.

SHOW PORTS

Displays those portions of the port descriptor table (PDT) that are port independent. Port-independent items in the PDT are used by all system communications services (SCS) port drivers.

FORMAT

SHOW PORTS

command parameters

None.

command qualifiers

/ADDRESS=n

Displays a specific port descriptor table, as specified by the address. The PDT summary page, which lists the addresses for individual PDTs, is the first display shown for the SHOW PORTS command.

DESCRIPTION

The SHOW PORTS command provides information for those CI ports with full SCS connections. Therefore, information about UDA ports and similar controllers is not included in the output of this command.

The initial display is a PDT summary page, listing the PDT address, port type, device type, and driver name for each PDT. Subsequent displays provide detailed information for each PDT listed on the summary page.

Information for a particular PDT can also be obtained by using the /ADDRESS qualifier to the SHOW PORTS command.

EXAMPLE

```
SDA> SHOW PORTS

                    --- PDT Summary Page ---
 PDT Address          Type         Device          Driver Name
 -----------          ----         -------         -----------
  801CEA20             pa           PAB0            PADRIVER

              ============================================

                 --- Port Descriptor Table (PDT) 801CEA20 ---

Type: 01 pa
Characteristics: 0000

Msg Header Size          32  Connect         8023222A  Recyclh_Msg_Buf  802323E7
Max Xfer Bcnt       FFFFFFFF  Dealloc_Dg_Buf  8023250C  Request_Data     80232677
DG Header Size           72  Disconnect      802322F3  Send_Data        802326BE
Poller Sweep             25  Unmap           80232730  Send_Dg_Buf      80232570
Fork Block W.Q.       empty  Map             802325D2  Send_Msg_Buf     8023248A
UCB Address          80335320  Map_Bypass     802325B9  Send_Cnt_Msg_Buf 80232491
ADP Address          00000000  Map_Irp        802325C2  Read_Count       8023279E
Accept               80232277  Map_Irp_Bypass 802325B1  Rls_Read_Count   802327DF
Alloc_Dg_Buf         802324F8  Queue_Dg_Buf   80232512  Mreset           802327E8
Alloc_Msg_Buf        80232383  Queue_Mult_Dgs 8023251A  Mstart           802327F0
Dealloc_Msg_Buf      8023243C  Recycl_Msg_Buf 802323F1  Stop_Vcs         8023281F
Dealloc_Msg_Buf_Reg 8023244F  Reject          802322DA  Send_Dg_Reg      80232563
```

SHOW PROCESS

Displays the software and hardware context of any process in the balance set and performs an implicit SET PROCESS command.

FORMAT

SHOW PROCESS *[/qualifier[,...]] [parameter]*

command parameters

ALL

Shows information about all the processes that exist in the system.

name

The name of the process that you want to see. Do not use this parameter with the /SYSTEM or /INDEX= qualifiers. The name can contain up to 15 letters and numerals and can include the underscore (_) and dollar sign ($) characters. The name must be a quoted string if it contains other than the foregoing characters.

If you specify no parameter, this command displays information on the current process. See the description of the SET PROCESS command for the definition of the current process.

command qualifiers

/ALL

Displays the information shown by the following qualifiers:

- /CHANNEL
- /LOCKS
- /PCB
- /PHD
- /REGISTERS
- /RMS
- /WORKING_SET
- /PROCESS_SECTION_TABLE
- /PAGE_TABLES

/CHANNEL

Displays the I/O channels assigned to the process, the address of the window-control block associated with that channel, the status of the channel, and the specification of the file or device associated with the channel.

The display produced by the SHOW PROCESS/CHANNEL command contains four columns of information, labeled "channel," "window," "status," and "device/file accessed." These columns contain the following information.

Column	Contents
Channel	The number of each channel assigned to the process
Window	The address of the window-control block for the file, if the device is a file-oriented device, zero otherwise
Status	The status of the device: "busy" if the device has an I/O operation outstanding, blank otherwise
Device/file accessed	The name of the device and, if applicable, the name of the file being accessed on that device

The display varies with the process chosen. SDA displays the information in the form

dcuu:(file-id)filename

where:

- **dcuu:** is the name of the device.
- **file-id** is the RMS file identification.
- **filename** is the full file specification, including directory name.

SDA displays some or all of this information under the following conditions:

Information Displayed	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:filename	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 file name.
dcuu:(file-id)filename	SDA displays this information only when you are examining a dump. The file name corresponds to the file-id on the device listed. If you are examining a dump from your own system, the file name is probably valid. If you are examining a dump from another system, the file name is probably meaningless in the context of your system.
dcuu:(file-id)	The file-id no longer points to a valid file name, 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 file name

/INDEX=nn

The index of the software process control block (PCB) into the system's PCB list. Alternatively, this argument can be the process identification (PID or EPID), from which SDA extracts the index. This index identifies the process to be displayed.

/LOCKS
Displays the locks owned by the current process. See the description of the SHOW RESOURCE command and its /LOCKID qualifier for a description of the information displayed by this qualifier.

/P0
Displays the page tables for P0 space; must be used with the /PAGE_TABLE qualifier.

/P1
Displays the page tables for P1 space; must be used with the /PAGE_TABLE qualifier.

/PAGE_TABLES [range]
Displays the page tables of the program and control regions. This qualifier produces a display in the same format as that of the SHOW PAGE_TABLE command.

The argument **range** can have two forms. When you provide an argument of the form x:y, where x and y are the addresses of virtual pages, this qualifier displays the page table entries that correspond to the range of pages from x to y.

When you provide an argument of the form x;y, where x and y are the addresses of virtual pages, this qualifier displays the page table entries that correspond to a range of y pages, starting with page x.

/PCB
Produces a list of the data contained in the software process control block (PCB). The software PCB is the central control mechanism for process swapping and scheduling.

This qualifier is the default.

The display produced by using the /PCB qualifier lists the following information:

- Software context for the process
- Condition-handling information
- Information on interprocess communication
- Information on counts, quotas, and resource usage

/PHD
Lists information included in the process header. The process header contains the vital statistics of a process and is swapped into memory when a process becomes part of the balance set. Each item listed by the PHD qualifier gives a quantity, count, or limit for the process concerning the following resources:

- Process memory
- The pager
- The scheduler
- Asynchronous system traps

- I/O activity
- CPU activity

/PROCESS_SECTION_TABLE

Lists the information contained in the process section table. The process section table contains entries, each of which describes a process section. The display that this qualifier produces is 132 columns wide.

SDA displays the offsets to the first and last entries in the process section table under the heading "process section table information," and then displays the contents of process section table. The following table describes the parts of each process section table entry contained in the display:

Part	Definition
INDEX	The offset into the section table of the process 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	The virtual address that marks the beginning of the first page of the section described by this entry.
PAGES	The 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	The cluster size used when faulting pages into this process section.
REFCNT	The 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	The flags that describe the access that processes have to the process section.

/REGISTERS

Lists the hardware context of the process. When a process executes, its hardware context is contained in the processor registers (see the description of the SHOW CRASH command). If the process is not executing, its hardware context is stored in the hardware PCB, which is part of the process header. The /REGISTERS qualifier displays the process registers in the following groups:

- General registers
- Stack pointers
- Special-purpose registers
- Base and length registers

If the process is the current process in a dump, the current registers are also displayed.

/RMS[=option]

Displays the RMS data structures for each image-I/O file the process has open.

If you provide the name of a structure as an option, this qualifier displays only the specified structure for each image-I/O file the process has open. If you do not specify an option, SDA displays the current list of options as determined by the last SET RMS command or by SDA when you start it.

See the description of the SET RMS command for the options you can use with this qualifier.

To show the RMS data structures for process-permanent files, use the following commands.

```
SDA> DEFINE SAVE=PIO$GW_IIOIMPA
SDA> DEFINE PIO$GW_IIOIMPA=PIO$GW_PIOIMPA
SDA> SHOW PROCESS/RMS

Process index: 003C   Name: WILLING   Extended PID: 22E0023C
------------------------------------------------------------
=============
IFAB Address: 7FFC5608          IFI: 0001       Organization: Sequential
=============                   ----            -------------

PRIMDEV:        08040007   REC,CCL,TRM,AVL,ODV
BKPBITS:        00090006   EOF,PPF_IMAGE,WRTACC,NORECLK
BLN:            2E            46.       BID:            0B            11.
EFN:            00                      MODE:           03
   .
   .
   .
```

/SYSTEM

Displays the system process control block. 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.

/WORKING_SET

Displays the working-set list for the process. The working-set list is a table that contains information for all virtual pages that the process can access without a page fault. This qualifier displays the follwing 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	The virtual address of the page, in the process address space, for which this entry exists
STATUS	A three-part section that lists the location of the page in physical memory, the type of page (see the description of the SHOW PAGE_TABLE command), and whether the page is locked into the working set

When SDA locates an unused working-set entry, it issues the message:

```
---- n empty entries
```

The value of n is the decimal number of contiguous, unused entries that SDA has found.

EXAMPLES

1

```
SDA> SHOW PROCESS/CHANNEL
Process index: 001C   Name: WILLING   Extended PID: 0000009C
-------------------------------------------------------------

                          Process active channels
                          -----------------------

Channel   Window        Status  Device/file accessed
-------   ------        ------  --------------------
  0010    00000000              MBA51:
  0020    801D3D00              DBA0:[SYS0.SYSEXE]SDA.EXE;1
  0030    00000000              DBA0:
  0040    00000000              DBA0:
  0050    00000000              MBA44:
  0060    00000000              MBA53:
  0070    801B8F40              DBA0:[SYS0.001001]LBRSHR.EXE;1 (section file)
  0080    801B8E20              DBA0:[SYS0.001001]DCXSHR.EXE;1 (section file)
  0090    00000000              TTE6:
  00A0    801B9060              DBA0:[SYS0.001001]LIBRTL.EXE;1 (section file)
  00B0    801B8D00              DBA0:[SYS0.001001]SCRSHR.EXE;1 (section file)
  00C0    00000000              TTE6:
  00D0    801D3A00              DRB0:[WILLING.BOOKS]SDAOUT.TXT;2
  00E0    00000000              TTE6:
```

The preceding example shows the output of a SHOW PROCESS/CHANNEL command used while analyzing a running system. The display includes the names of the files that are opened because the process in which SDA is running has access to the files. If the process has no access to a file, SDA displays only the device name, the file identification (FID), and the directory name.

2

```
SDA> SHOW PROCESS/CHANNEL
Process index: 0008   Name: JNLACP   Extended PID: 00000088
-------------------------------------------------------------

                          Process active channels
                          -----------------------

Channel   Window        Status  Device/file accessed
-------   ------        ------  --------------------
  0010    801B11A0              DBA0:(32,1,0)[SYS0.SYSEXE]F11BXQP.EXE;1 (section file)
  0020    801B11A0              DBA0:(32,1,0)[SYS0.SYSEXE]F11BXQP.EXE;1 (section file)
  0030    00000000              DBA0:
  0040    00000000              DBA0:
  0050    00000000              MBA11:
  0060    801BD6E0              DBA0:(252,1,0)[SYS0.SYSEXE]JNLACP.EXE;1
  0070    801B8C40              DBA0:(109,1,0)[SYS0.001001]LIBRTL.EXE;1 (section file)
  0080    801B6960              DBA0:(99,1,0)[SYS0.001001]DISMNTSHR.EXE;1 (section file)
  0090    801B6AE0              DBA0:(100,1,0)[SYS0.001001]MOUNTSHR.EXE;1 (section file)
  00A0    801B1AA0              DRA2:(18,36,0)
  00B0    8016D380              DRB0:(8757,12,2)[JOURNAL]JWHBI.BIJ;1
```

The preceding example shows the display produced by the SHOW PROCESS /CHANNEL command while SDA is analyzing a crash dump. Note that each file specification in the display includes the file identification (FID).

3
```
SDA> SHOW PROCESS/INDEX=47
Process index: 0047   Name: C_EMACS   Extended PID: 00000447
------------------------------------------------------------

Process status:  00040001   RES,PHDRES

PCB address              8011AD70    JIB address              801127F0
PHD address              80606C00    Swapfile disk address    02002821
Master internal PID      00070040    Subprocess count                1
Internal PID             00080047    Creator internal PID     00070040
Extended PID             00000447    Creator extended PID     000003C0
State                       HIB      Termination mailbox          06D8
Current priority              5      AST's enabled                KESU
Base priority                 4      AST's active                 NONE
UIC                     [010,115]    AST's remaining                13
Mutex count                   0      Buffered I/O count/limit        6/6
Waiting EF cluster            0      Direct I/O count/limit          6/6
Starting wait time       00001B1A    BUFIO byte count/limit     11164/11420
Event flag wait mask     FFBFFFFF    # open files allowed left      18
Local EF cluster 0       C8600C05    Timer entries allowed left     10
Local EF cluster 1       00000000    Active page table count         0
Global cluster 2 pointer 00000000    Process WS page count          78
Global cluster 3 pointer 00000000    Global WS page count           33
```

The preceding example shows the output of a SHOW PROCESS/INDEX command.

4
```
SDA> SHOW PROCESS/RMS=IFAB
VAX/VMS 4.4 -- System Dump Analysis     06-JAN-1986 16:12:02.86    Page 1
                                                Table of Contents
                                                -----------------
                                    Process index: 000F   Name: NETA ....   2
VAX/VMS 4.4 -- System Dump Analysis    06-JAN-1986 16:12:02.86
Page 2
Process index: 000F   Name: NETACP   Extended PID: 21C0010F
=============
IFAB Address: 7FFA9208    IFI: 0001    Organization: Sequential
=============             ----         -------------

PRIMDEV:        1C4D4008   DIR,FOD,SHR,AVL,ELG,IDV,ODV,RND
BKPBITS:        02080000   NORECLK,SEARCH
BLN:            2E         46.          BID:            0B          11.
EFN:            00                      MODE:           00
IOS:            00000000                ASBADDR:        00000000
IOS4:           00000000                ARGLST:         7FFE7D28
IRAB_LNK:       00000000                CHNL:           0060
FAC:            00
ORGCASE:        00         Sequential
LAST_FAB:       0000A8E4                NWA_PTR:        00000000
IFI:            0001                    ECHO_ISI:       0000
JNLBDB:         00000000                FWA_PTR:        7FFA9400
BDB_FLNK:       7FFA9248                DEVBUFSIZ:      0200        512.
BDB_BLNK:       7FFA9248                RTDEQ:          0000        0.
RFMORG:         00         UDF
RAT:            00
LRL:            0000       0.           HBK_DISK:       00000000
FFB:            0000       0.           EBK_DISK:       00000000
FSZ:            00         0.           BKS:            00          0.
DEQ:            0000       0.           MRS:            0000        0.
HBK:            00000000   0.           GBC:            0000        0.
EBK:            00000000   0.
RNS_LEN:        00000000                LOCK_BDB:       00000000
SFSB_PTR:       00000000                AVLCL:          0000        0.
GBSB_PTR:       00000000                AVGBPB:         0000        0.
GBH_PTR:        00000000                RJB_PTR:        00000000
```

```
JNLFLGS:        00
RECVRFLGS:      00
JNLFLGS2:       00
EXTJNL_PTR:     00000000                PAR_LOCK_ID:    00000000
BLBFLNK:        00000000
BLBBLNK:        00000000
AS_DEV:         1C4D4008    DIR,FOD,SHR,AVL,ELG,IDV,ODV,RND
ASDEVBSIZ:      0200        512.
```

The preceding example shows the output of the SHOW PROCESS /RMS=IFAB command.

SHOW RESOURCE

Displays information on system resources.

FORMAT

SHOW RESOURCE *[/qualifier]*

command parameters

None.

command qualifiers

/ALL

Displays information on all the locks in the system.

/LOCKID=nn

Displays information on the resource associated with the lock with identification nn.

DESCRIPTION

The SHOW RESOURCE command displays information on system resources. The following sections discuss each of the displays this command produces.

SHOW RESOURCE/ALL

The display produced by the SHOW RESOURCE/ALL command contains information as shown in the example following. Descriptions of each item in the display follow the example. This qualifier is the default.

```
SDA> SHOW RESOURCE/ALL

Resource database
-----------------
Address of RSB:  801FCB40  Group grant mode:       PW
Parent RSB:      00000000  Conversion grant mode:  CR
Sub-RSB count:          0  BLKAST count:            0
Value block:   00000000 00000000 00000000 00000000
Resource:      454F5024 4B534944    DISK$POE
 Length    8   00000000 00000000    ........
 User mode     00000000 00000000    ........
 Group   360   00000000 00000000    ........
Granted queue (Lock ID / Gr mode):
     06AB010C  CR
Conversion queue (Lock ID / Gr/Rq mode):
     095B00F2  PW/EX
Waiting queue (Lock ID / Rq mode):
     054400BC  EX
    .
    .
    .
```

The definitions of the fields in the display are as follows:

Field	Contents
Address of RSB	The address of the resource block that describes this resource.
Parent RSB	The resource block that is the parent of this resource block. This field is 0 in the example, which means that this resource block is a parent block (is not a child block).
Sub-RSB count	The number of RSBs of which this RSB is the parent. This RSB has no "children."
Group grant mode	The most restrictive mode in which a lock on this resource has been granted. The values that this field can contain are listed following and shown in order from the least restrictive to most restrictive lock modes. The most restrictive lock granted on this resource is in protected-write mode. For information on how the lock modes can conflict with each other, see the *VAX/VMS System Services Reference Manual.*

Value	Meaning
NL	Null mode
CR	Concurrent-read mode
CW	Concurrent-write mode
PR	Protected-read mode
PW	Protected-write mode
EX	Exclusive mode

Conversion grant mode	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 lock at the head of the conversion queue is waiting.
BLKAST count	The number of locks on this resource that have requested a blocking AST.
Value block	A hexadecimal dump of the 16-byte block value block associated with this resource.
Resource	The beginning of the three-column dump of the name of this resource, which is stored in the resource block. The first two columns are the hexadecimal representation of the name, with the least significant byte represented by the rightmost two digits in the righthand 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 unprintable ASCII characters.
Length	The length, in bytes, of the resource block.

Field	Contents
User mode	The processor mode of the name space in which this resource block resides. In the preceding example, the resource block resides in the user-mode name space. There is a name space for each processor-access mode.
System	The owner of the resource. In this example the owner is the system, the VAX/VMS operating system. When the owner of the resource is a group, this field contains the number (in octal of the owning group).
Granted queue	The list of locks on this resource that have been granted. For each lock in the list, this display contains the number of the lock and the lock mode in which the lock was granted. In the previously shown example, lock number 06AB010C was granted in common-read mode.
Conversion queue	The list of locks waiting to be converted from one mode to another. For each lock in the list, this display contains the number of the lock, the mode in which the lock was granted, and the mode to which the lock is to be converted. In the previously shown example, lock number 095B00F2 is waiting to be converted from protected-write mode to exclusive-access mode.
Waiting queue	The list of locks waiting to be granted. For each lock in the list, this display contains the number of the lock and the mode requested for that lock. In the previously shown example, lock number 054400BC is waiting to be granted in exclusive-access mode.

SHOW RESOURCE/LOCKID

The SHOW RESOURCE/LOCKID command shows information on the resource locked by the lock identified by the parameter to the /LOCKID qualifier. The following example shows the information contained in the display produced by this qualifier. A description of each item in the display follows the example.

```
SDA> SHOW PROCESS/LOCKS/INDEX=8

Process index: 0008   Name: JOB_CONTROL   Extended PID: 22A00088
------------------------------------------------------------------
Lock data:

Lock id:  11A00117   PID:    00010008   Flags:   CONVERT SYNCSTS SYSTEM
Par. id:  007B0004   Granted at    NL
Sublocks:        0
LKB:      801F9B40
Resource:       00000000 0000018C    ........  Status:
 Length   04    00000000 00000000    ........
 Exec. mode     00000000 00000000    ........
 System         00000000 00000000    ........
```

```
Lock id:  007B0004   PID:    00010008   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  00000000   Granted at    CR            SYSTEM
Sublocks:       16
LKB:      801E9100
Resource:       00000014 03BA0001     ........  Status:
 Length   19   44244854 4E594C50     PLYNTH$D
 Exec. mode     00000000 00304155     UA0.....
 System         00000000 00000000     ........
```

The information in the display is described as follows:

Display Element	Description
Process Index	The index into the PCB array at which you find a pointer to the PCB of the process that owns the lock.
Name	The name of the process that owns the lock.
Extended PID	The clusterwide identification of the process that owns this lock.
Lock Data	The heading of the section of the display that contains the information on the lock.
Lock id	The identification of the lock.
PID	The processor-wide identification of the lock.
FLAGS	Information specified in the request for the lock.
Par id	The identification of this lock's parent lock.
Granted at	The lock mode at which this lock was granted.
Sublocks	The identification numbers of the locks that this lock owns.
LKB	The address of the lock block, a block of memory in nonpaged dynamic pool in which the information on this lock is stored.
Resource	A dump of the resource block. The two lefthand columns show the contents of the resource block as hexadecimal values, the least significant byte being represented by the rightmost two digits. The righthand column shows the contents of the resource block as ASCII text, the least significant byte being represented by the leftmost character.
Length	The length, in bytes, of the lock block.
System	The owner of the lock. This lock is owned by the VAX/VMS operating system. Locks owned by a group have the number of the owning group in this field.
Status	The status of the lock, information used internally by the VAX/VMS lock manager.

The two columns of hexadecimal numbers are a dump of the lock block. The column of ASCII characters to the right of the hexadecimal columns is the ASCII representation of the dump of the lock block.

SHOW RMS

Displays the names of the VAX RMS data structures that the SHOW PROCESS/RMS command displays.

FORMAT

SHOW RMS

command parameters

None.

command qualifiers

None.

DESCRIPTION

This command shows names of the data structures that the SHOW PROCESS /RMS command displays. The SET RMS command determines which data structures SHOW PROCESS/RMS displays. See the description of the SET RMS command for the names of the VAX RMS data structures that this command can display.

EXAMPLE

```
SDA> SHOW RMS
RMS Display Options: IFB,IRB,IDX,BDB,BDBSUM,ASB,CCB,WCB,FCB,FAB,RAB,
NAM,XAB,RLB,BLB,BLBSUM,GBD,GBH,TRC,FWA,GBDSUM,RJB
Display RMS structures for all IFI values.
```

The preceding example shows how the SHOW RMS command displays the data structures to be displayed by the SHOW PROCESS/RMS command.

SHOW RSPID

Displays information about response-IDs (RSPIDs). Whenever a local system application (SYSAP) requires a response from a remote SYSAP, a unique number is assigned by the local system and is called an RSPID. 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.

FORMAT

SHOW RSPID

command parameters

None.

command qualifiers

/CONNECTION=n

Displays RSPID information for the specific SCS connection whose connection descriptor table (CDT) address is given. The value that you specify to obtain information for a specific CDT is obtained either by using the SHOW CONNECTIONS command or by examining an appropriate data structure.

DESCRIPTION

The SHOW RSPID command displays information about the response-ID descriptor table, which lists the currently open requests that require responses from remote SYSAPs. For each request that a local SYSAP transmits that requires a response from a remote SYSAP, VAX/VMS generates a unique identifying number that is assigned to that request, and that identifying number is called a response-ID (RSPID).VAX/VMS can associate a specific RSPID with the request that caused the RSPID to be generated, and the SHOW RSPID command provides information about RSPIDs that are currently in use (in other words, currently outstanding requests from local SYSAPs to remote SYSAPs).

The display for the SHOW RSPID command, shown in the following example, lists a summary of the current response descriptor table. For each RSPID, this summary includes the following:

- The RSPID value
- The address of the class driver request packet (CDRP, which generally represents the original request)
- The address of the CDT that is using the RSPID
- The name of the local process using the RSPID
- The remote node from which a response is required (and has not been received).

To obtain information about RSPIDs in use for a specific connection, use the /CONNECTION= qualifier and the associated CDT address.

EXAMPLE

```
SDA> SHOW RSPID
        --- Summary of Response Descriptor Table (RDT) 801C2108 ---

RSPID     CDRP Address  CDT Address  Local Process Name  Remote Node
-----     ------------  -----------  ------------------  -----------

F8680001   80336B00      801C2850    VMS$DISK_CL_DRVR    PINTO
F8070002   80422A00      801C28F0    VMS$VAXcluster      SOLLY
F80D0003   80393C60      801C2850    VMS$DISK_CL_DRVR    PINTO
```

SHOW STACK

Displays the location and contents of the four process stacks and the systemwide interrupt stack.

FORMAT

SHOW STACK *[/qualifier[,...]] [range]*

command parameter

range

The range of memory locations you want to display in stack format. You can express the range as two locations separated by a colon (:), or as a location and a length, in bytes, separated by a semicolon (;).

command qualifiers

/EXECUTIVE

Shows the executive-mode stack for the current process.

/INTERRUPT

Shows the interrupt-mode stack.

/KERNEL

Shows the kernel-mode stack for the current process.

/SUPERVISOR

Shows the supervisor-mode stack for the current process.

/USER

Shows the user-mode stack for the current process.

DESCRIPTION

Each qualifier displays one of four stacks that correspond to the four VAX/VMS processor access modes for the current process as specified by the most recent SET PROCESS or SHOW PROCESS command. The /INTERRUPT qualifier displays the systemwide interrupt stack. The default for SHOW STACK is the stack that is currently being used or that was in use when the system failed.

The following example shows the display produced by the SHOW STACK command. The display is the same for each stack and is composed of the following four sections.

Section	Contents
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 next to the stack addresses.
Symbols	SDA attempts to display the contents of a location symbolically, using a symbol and an offset, as shown in the example following. If the address is not within FFF (hexadecimal) of the value of any existing symbol, the field is left blank.

If a stack is empty, the display shows:

```
SP =>              (THE STACK IS EMPTY)
```

If you give a range of memory locations as a parameter to the SHOW STACK command, SDA displays that range in stack format.

EXAMPLE

```
SDA> SHOW STACK
Current operating stack
-----------------------
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 preceding example shows the output from a SHOW STACK command. The data shown above the stack pointer may 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 these processes.

FORMAT

SHOW SUMMARY

command parameters

None.

command qualifier

/IMAGE

This qualifier causes SDA to provide an extra line for every process in the display that SHOW SUMMARY produces. The line contains the name of the image, if available, being executed within each process.

DESCRIPTION

The following example shows the display produced by the SHOW SUMMARY command.

Column	Contents
Indx	The index of this process into the PCB array
Extended PID	The 32-bit number that uniquely identifies the process
Process name	The name assigned to the process
Username	The name of the user who created the process

Column	Contents	
State	The current state of the process, one of 14 states	
	State	**Meaning**
	COM	Computable and resident in memory
	COMO	Computable, but outswapped
	CUR	Currently executing
	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	The current scheduling priority of the process	
PCB	The address of the control block for this process	
PHD	The address of the header for this process	
Wkset	The number (decimal) of pages currently in the working set of the process	

EXAMPLE

```
SDA> SHOW SUMMARY
 Extended Indx Process name Username State Pri   PCB      PHD     Wkset
-- PID -- ---- ------------ -------- ----- --- -------- -------- -----
 00000080 0000 NULL                   COM    0 80001EA0 80001D28     0
 00000081 0001 SWAPPER                HIB   16 80002168 80001FF0     0
 00000083 0003 DBAOBACP      SYSTEM   HIB   10 800DD330 80202400   469
 00000087 0007 OPCOM         SYSTEM   LEF    7 800DDB90 80332200    90
    .
    .
    .
```

The SHOW SUMMARY command displays a summary description of all the processes that existed on the system at the time of the system failure. The display is the same as that produced by the DCL command SHOW SYSTEM.

SHOW SYMBOL

Displays the value of a symbol and the contents of the location that has an address equal to the value of that symbol.

FORMAT

SHOW SYMBOL *[/qualifier[,...]] parameter*

command parameter

symbol-name

The name of the symbol of interest. You must provide the name of the symbol.

command qualifier

/ALL

Displays information on all the symbols whose names match the characters specified in the command parameter and are defined in the SDA symbol table. If you use this qualifier, you can provide part of the name of a symbol, and SDA will display all symbols that start with the string you specify.

EXAMPLES

1
```
SDA> SHOW SYMBOL G
G = 80000000 :  8FBC0FFC
```

This command shows two items: G has a value of 80000000, and at address 80000000 is the quantity 8FBC0FFC (hexadecimal).

2
```
SDA> SHOW SYMBOL/ALL IOC$GL_
Symbols sorted by name
----------------------
IOC$GL_ADPLIST  80000E58 => 801F4000
IOC$GL_AQBLIST  800027B0 => 8020C080
IOC$GL_CRBTMOUT 800027C4 => 801F5594
IOC$GL_DEVLIST  80000EBC => 80000ECC
    .
    .
    .
IOC$GL_SRPSIZE  80002764 => 00000080
IOC$GL_SRPSPLIT 8000276C => 801F4000
IOC$GL_TU_CDDB  800027CC => 00000000
```

The preceding example shows the display produced by the SHOW SYMBOL /ALL command. SDA searches its symbol table for all symbols that begin with the string "IOC$GL_" and displays the symbols, their values, and the contents of the locations addressed by those values.

SPAWN

Creates a subprocess of the current process. The system copies the context of the subprocess from the current process. The subprocess executes the command that you specify in the SPAWN command.

FORMAT

SPAWN *[/qualifier[,...]] [command]*

command parameter

command

The name of the command that you want executed by the subprocess.

command 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.

/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 in order 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.

VALIDATE QUEUE

Validates the integrity of the specified queue by checking the pointers in the queue.

FORMAT

VALIDATE QUEUE *[address]*

command parameter

address

The address is the address of an element of the queue. If the address is not specified, then the last element address that was specified (in a previous VALIDATE QUEUE command) is used. If a dot is specified as the address, then the last evaluated expression is used as the address of an element of the queue.

command qualifiers

/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 the pointers are all 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
```

If SDA discovers an error in the queue, it displays this message:

```
Error comparing backward link to previous structure address
mmmmmmmm.  Error occurred in queue element at address nnnnnnnn,
after tracing n elements.
```

Address mmmmmmmm is the address you gave as the argument to the VALIDATE QUEUE command; address nnnnnnnn is the address of the queue element in which the error was detected.

If there are no entries in the queue, SDA displays this message:

```
The queue is empty
```

If you do not specify an address to the VALIDATE QUEUE command, the following action is taken:

- If a queue has previously been used in the current SDA session, that queue is used.
- The previous evaluated expression is used as an address if you specify a period (.) after the VALIDATE QUEUE command.
- If no queue has previously been specified, and you do not use a period after the command, an error message is displayed indicating that no queue has been specified for validation.

EXAMPLE

```
SDA> VALIDATE QUEUE IOC$GL_IRPFL
Error comparing backward link to previous structure address (801E6200)
Error occured in queue element at address 801CA000, after tracing 399 elements
SDA> VALIDATE QUEUE IOC$GL_IRPFL
Queue is complete, total of 404 elements in the queue
```

The preceding example shows the display produced by the VALIDATE QUEUE command. The preceding example was produced on a running system, and the queue changed during the SDA attempt to validate it. Thus the error message. The second command was successful, and the queue was complete.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

VAX/VMS System Dump Analyzer
Reference Manual
AA–Z429C–TE

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

Cut Along Dotted Line

VAX/VMS System Dump Analyzer
Reference Manual
AA–Z429C–TE

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here

Cut Along Dotted Line