OpenVMS Alpha System Dump Analyzer Utility Manual

Order Number: AA–PV6UC–TE

November 1996

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

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

Software Version: OpenVMS Alpha Version 7.1

Digital Equipment Corporation
Maynard, Massachusetts

November 1996

ZK6135

The OpenVMS documentation set is available on CD–ROM.

Contents

Figures

Tables

Preface

Intended Audience

The *OpenVMS Alpha System Dump Analyzer Utility Manual* is intended primarily for the system programmer who must investigate the causes of system failures and debug kernel mode code, such as a device driver. An understanding of data structures is necessary to accurately interpret the results of System Dump Analyzer (SDA) commands.

This manual also includes such system management information as maintaining the system resources necessary to capture and store system crash dumps. If you need to determine the cause of a hung process or improve system performance, refer to this manual for instructions on using SDA to analyze a running system.

Document Structure

The *OpenVMS Alpha System Dump Analyzer Utility Manual* includes the following information:

- An introduction to the functions, features, and key concepts of the System Dump Analyzer (SDA). This part also includes instructions for maintaining the optimal environment to analyze system failures.
- Instructions about how to:
  - Invoke SDA.
  - Exit from SDA.
  - Record the output of an SDA session.
- A description of those qualifiers to the ANALYZE command that govern the behavior of SDA.
- A description of the function, format, and parameters of each SDA command. It also provides usage examples for each command.

Related Documents

For additional information, refer to the following documents:

- *OpenVMS Alpha Version 7.1 Upgrade and Installation Manual*
- *OpenVMS Calling Standard*
- *OpenVMS System Manager's Manual: Essentials*
- *OpenVMS System Manager's Manual: Tuning, Monitoring, and Complex Systems*
- *OpenVMS Programming Interfaces: Calling a System Routine*
- *Writing OpenVMS Alpha Device Drivers in C*

- *OpenVMS AXP Internals and Data Structures*
- *Alpha Architecture Reference Manual*
- *MACRO–64 Assembler for OpenVMS AXP Systems Reference Manual*

For additional information on OpenVMS products and services, access the Digital OpenVMS World Wide Web site. Use the following URL:

`http://www.openvms.digital.com`

Reader's Comments

Digital welcomes your comments on this manual.

Print or edit the online form SYS$HELP:OPENVMSDOC_COMMENTS.TXT and send us your comments by:

Internet	**openvmsdoc@zko.mts.dec.com**
Fax	603 881-0120, Attention: OpenVMS Documentation, ZK03-4/U08
Mail	OpenVMS Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How To Order Additional Documentation

Use the following table to order additional documentation or information. If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Location	Call	Fax	Write
U.S.A.	DECdirect 800−DIGITAL 800−344−4825	Fax: 800−234−2298	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	809−781−0505	Fax: 809−749−8300	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street, Suite 200 P.O. Box 11038 Metro Office Park San Juan, Puerto Rico 00910−2138
Canada	800−267−6215	Fax: 613−592−1946	Digital Equipment of Canada, Ltd. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	—	—	Local Digital subsidiary or approved distributor
Internal Orders	DTN: 264−4446 603−884−4446	Fax: 603−884−3960	U.S. Software Supply Business Digital Equipment Corporation 8 Cotton Road Nashua, NH 03063−1260

ZK−7654A−GE

Conventions

The name of the OpenVMS AXP operating system has been changed to OpenVMS Alpha. Any references to OpenVMS AXP or AXP are synonymous with OpenVMS Alpha or Alpha.

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

The following conventions are also used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x* or GOLD *x*	A sequence such as PF1 *x* or GOLD *x* indicates that you must first press and release the key labeled PF1 or GOLD and then press and release another key or a pointing device button. GOLD key sequences can also have a slash (/), dash (–), or underscore (_) as a delimiter in EVE commands.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	Horizontal ellipsis points in examples indicate one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	Vertical ellipsis points indicate the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command format descriptions, brackets indicate optional elements. You can choose one, none, or more than one of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification or in the syntax of a substring specification in an assignment statement.)
[|]	In command format descriptions, vertical bars separating items inside brackets indicate that you choose one, none, or more than one of the options.
{ }	In command format descriptions, braces indicate a required choice of options; you must choose one of the options listed.

{ | }	In command format descriptions, vertical bars separating items inside braces indicate that you choose one item from among those listed. If you choose no items from among those listed, you in effect choose the default item, which is indicated by a (d) after it. However, if there is no default item, then you must choose one of the options listed.
text style	This text style represents the introduction of a new term or the name of an argument, an attribute, or a reason. This style is also used to show user input in Bookreader versions of the manual.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
Monospace type	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

SDA Description

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

You can use SDA commands to perform the following operations:

- Direct (or echo) the output of an SDA session to a file or device (SET OUTPUT or SET LOG).
- Display the condition of the operating system and the hardware context of each processor in the system at the time of the system failure (SHOW CRASH or CLUE CRASH).
- Select a specific processor in a multiprocessing system as the subject of analysis (SET CPU).
- Select the default size of address data manipulated by the EXAMINE and EVALUATE commands (SET FETCH).
- Enable or disable the sign extension of 32-bit addresses (SET SIGN_ EXTEND).
- Display the contents of a specific process stack (SHOW STACK or CLUE STACK).
- Format a call frame from a stack location (SHOW CALL_FRAME).
- Read a set of global symbols into the SDA symbol table (READ).
- Define symbols to represent values or locations in memory and add them to the SDA symbol table (DEFINE).
- Evaluate an expression in hexadecimal and decimal, interpreting its value as a symbol, a condition value, a page table entry (PTE), or a processor status (PS) quadword (EVALUATE).
- Examine the contents of memory locations, optionally interpreting them as Alpha assembler instructions, a PTE, or a PS (EXAMINE).
- Display device status as reflected in system data structures (SHOW DEVICE).
- Display the contents of the stored machine check frame (SHOW MACHINE_ CHECK or CLUE MCHK) for selected Digital computers.
- Format system data structures (FORMAT).
- Validate the integrity of the links in a queue (VALIDATE QUEUE).
- Display a summary of all processes on the system (SHOW SUMMARY).
- Show the hardware or software context of a process (SHOW PROCESS or CLUE PROCESS).
- Display the OpenVMS RMS data structures of a process (SHOW PROCESS with the /RMS qualifier).
- Display memory management data structures (SHOW POOL, SHOW PFN_DATA, SHOW PAGE_TABLE, or CLUE MEMORY).

- Display lock management data structures (SHOW RESOURCE or SHOW LOCK).
- Display OpenVMS Cluster management data structures (SHOW CLUSTER, SHOW CONNECTIONS, SHOW RSPID, or SHOW PORTS).
- Display multiprocessor synchronization information (SHOW SPINLOCKS).
- Display the layout of the executive images (SHOW EXECUTIVE).
- Capture and archive a summary of dump file information in a list file (CLUE HISTORY).
- Copy the system dump file (COPY).
- Define keys to invoke SDA commands (DEFINE/KEY).
- Search memory for a given value (SEARCH).

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

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

Caution:

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

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

1 System Management and SDA

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

1.1 Writing System Dumps

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

- Have the system dump file written to either SYSDUMP.DMP (the system dump file) or to PAGEFILE.SYS (the primary system page file).

- Set the DUMPSTYLE system parameter to 0 or 2 (for dumps containing all physical memory) or to 1 or 3 (for dumps containing only selected virtual addresses).

See Section 1.1.1 for more information about the DUMPSTYLE parameter values.

1.1.1 Dump File Style

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

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

Table SDA–1 The DUMPSTYLE Mask

Bit	Value	Description
0	0	0= Full dump (SYSGEN default). The entire contents of physical memory will be written to the dump file.
		1= Selective dump. The contents of memory will be written to the dump file selectively to maximize the usefulness of the dump file while conserving disk space.
1	2	0= Minimal console output.
		1= Full console output (includes stack dump, register contents, and so on.)
2	4	This bit is ignored on Alpha systems.
3	8	0= Do not compress.
		1= Compress.
4-31		Reserved to Digital

In a physical memory dump, the DUMPSTYLE system parameter can be set to 0,2,8, or 10. Each value provides a full dump; the value of 0 yields an uncompressed dump with minimal console output; the value of 2 provides an uncompressed dump with full console output; the value of 8 provides a compressed dump with minimal console output; and the value of 10 provides a compressed dump with full console output. A physical memory dump requires that all physical memory be written to the dump file. This ensures the presence of all the page table pages required for SDA to emulate translation of system virtual addresses. These table pages include the level 1 page table of the current process, the shared level 2 page table that maps the system page table (SPT), and the level 3 page table pages that constitute the SPT.

In certain system configurations, it may be impossible to preserve the entire contents of memory in a disk file. For instance, a large memory system or a system with small disk capacity may not be able to supply enough disk space for a full memory dump. If the system dump file cannot accommodate all of memory, information essential to determining the cause of the system failure may be lost.

To preserve those portions of memory that contain information most useful in determining the causes of system failures, a system manager sets the value of the DUMPSTYLE system parameter to 1, 3, 9, or 11 to specify a dump of selected virtual address spaces. Each value provides a selective dump; the value of 1 yields an uncompressed dump with minimal console output; the value of 3 provides an uncompressed dump with full console output; the value of 9 provides a compressed with minimal console output; and the value of 11 provides a compressed with full console output. In a selective dump, related pages of virtual address space are written to the dump file as a unit called a logical memory block (LMB). For example, one LMB consists of the system and global page tables; another is the address space of a particular process. Those LMBs most likely to be useful in crash dump analysis are written first.

Table SDA–2 compares full and selective style dump files.

Table SDA–2 Comparison of Full and Selective Dump Files

Item	Full	Selective
Available Information	Complete contents of physical memory in use, stored in order of increasing physical address.	System page table, global page table, system space memory, and process and control regions (plus global pages) for all saved processes.
Unavailable Information	Contents of paged-out memory at the time of the system failure.	Contents of paged-out memory at the time of the system failure, process and control regions of unsaved processes, L1 page tables, and memory not mapped by a page table.
SDA Command Limitations	None.	The following commands are not useful for unsaved processes: SHOW PROCESS /CHANNELS, SHOW PROCESS/IMAGE, SHOW PROCESS/RMS, SHOW STACK, and SHOW SUMMARY/IMAGE.

1.1.2 Controlling the Size of Page Files and Dump Files

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

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

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

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

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

1.1.3 Writing to the System Dump File

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

SYS$SYSTEM:SYSDUMP.DMP (SYS$SPECIFIC:[SYSEXE]SYSDUMP.DMP) is furnished as an empty file in the OpenVMS Alpha software distribution kit. To successfully store a crash dump, SYS$SYSTEM:SYSDUMP.DMP must be enlarged to hold all of the page tables required for SDA to emulate system virtual address translation.

To calculate the correct size for a physical memory dump to SYS$SYSTEM:SYSDUMP.DMP, use the following formula:

```
size-in-blocks(SYS$SYSTEM:SYSDUMP.DMP)
        = size-in-pages(physical-memory) * blocks-per-page
        + number-of-error-log-buffers * blocks-per-buffer
        + 2
```

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

1.1.4 Writing to the Dump File off the System Disk

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

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

- The dump device directory structure must resemble the current system disk structure. The [SYSn.SYSEXE]SYSDUMP.DMP file will reside there, with the same boot time system root.

  You can use AUTOGEN to create this file. In the MODPARAMS.DAT file, the following symbol prompts AUTOGEN to create the file:

  ```
  DUMPFILE_DEVICE = $nnn$ddcuuuu
  ```

- The dump device cannot be part of a volume set. Digital also strongly recommends that the dump device not be part of a shadow set.

- The DUMP_DEV environment variable must exist on your system. You specify the dump device at the console prompt, using the following format:
  `>>>SET DUMP_DEV device-name[...]`

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

  By specifying an alternate path DUMP_DEV, the disk can fail over to the alternate path when the system is running. If the system crashes subsequently, the bug-check code can use the alternate path by referring to the contents of DUMP_DEV.

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

For information on how to write the system dump file to an alternative device to the system disk, see the *OpenVMS System Manager's Manual: Tuning, Monitoring, and Complex Systems*.

1.1.5 Writing to the System Page File

If SYS$SYSTEM:SYSDUMP.DMP does not exist, the operating system writes the dump of physical memory into SYS$SYSTEM:PAGEFILE.SYS, the primary system page file, overwriting the contents of that file.

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

To calculate the minimum size for a physical memory dump to SYS$SYSTEM:PAGEFILE.SYS, use the following formula:

```
size-in-blocks(SYS$SYSTEM:PAGEFILE.SYS)
        = size-in-pages(physical-memory) * blocks-per-page
        + number-of-error-log-buffers  * blocks-per-buffer
        + 2
        + value of the system parameter RSRVPAGCNT
```

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

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

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

- Include SDA commands that free dump space in the site-specific startup command procedure (described in Section 1.3).
- Use the SDA COPY command to copy the dump from SYS$SYSTEM:PAGEFILE.SYS to another file. Use the SDA COPY command instead of the DCL COPY command because the SDA COPY command causes the pages occupied by the dump to be freed from the system's page file.

- If you do not need to copy the dump elsewhere, issue an ANALYZE /CRASH_DUMP/RELEASE command. When you issue this command, SDA immediately releases the pages to be used for system paging, effectively deleting the dump. Note that this command does not allow you to analyze the dump before deleting it.

1.2 Saving System Dumps

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

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

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

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

1.3 Invoking SDA when Rebooting the System

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

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

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

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

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

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

```
!
! SDA command file, to be executed as part of the system
! bootstrap from within CLUE. Commands in this file can
! be used to save the dump file after a system bugcheck, and
! to execute any additional SDA commands.
!

! Note that the logical name DMP$ must have been defined
! within SYS$MANAGER:SYLOGICALS.COM
!
READ/EXEC                   ! read in the executive images' symbol tables
COPY DMP$:SAVEDUMP.DMP      ! copy and save dump file
SHOW STACK                  ! display the stack
!
```

The SDA commands in this site-specific command file are executed first and then the CLUE HISTORY command is executed by default. See the reference section on CLUE HISTORY for details on the summary information that is generated and stored in the CLUE list file by the CLUE HISTORY command.

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

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

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

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

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

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

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

2 Analyzing a System Dump

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

- Verifying that the process invoking it is suitably privileged to read the dump file
- Using RMS to read in pages from the dump file

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

For detailed information on investigating system failures, see Section 6.

2.1 Requirements

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

2.2 Invoking SDA

If your process can access the files listed in Section 2.1, you can issue the DCL command ANALYZE/CRASH_DUMP to invoke SDA. If you do not specify the name of a dump file in the command, SDA prompts you:

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

The default file specification is as follows:

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

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

If you are rebooting after a system failure, SDA is automatically invoked. See Section 1.3.

2.3 Mapping the Contents of the Dump File

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

- If SDA cannot find the SPT in the dump file, it displays the following message:

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

  If that error message is displayed, you cannot analyze the crash dump, but must take steps to ensure that any subsequent dump can be analyzed. To do this, you must adjust the DUMPSTYLE system parameter as discussed in Section 1.1.1 or increase the size of the dump file as indicated in Section 1.1.2.

- If SDA finds the SPT in an incomplete dump, the following message is displayed:

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

Under certain conditions, some memory locations might not be saved in the system dump file. Additionally, if a bugcheck occurs during system initialization, the contents of the register display may be unreliable. The symptom of such a bugcheck is a SHOW SUMMARY display that shows no processes or only the swapper process.

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

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

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

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

2.4 Building the SDA Symbol Table

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

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

```
OpenVMS Alpha System Dump Analyzer

Dump taken on 27-MAR-1993  11:22:33.92
BADPAGFILD, Bad page file address deallocated
```

2.5 Executing the SDA Initialization File (SDA$INIT)

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

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

```
READ SDA$READ_DIR:filename
```

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

```
READ/EXECUTIVE SDA$READ_DIR:
```

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

```
SDA>
```

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

An SDA initialization file may invoke a command procedure with the @ command. However, such command procedures cannot invoke other command procedures.

3 Analyzing a Running System

Occasionally, OpenVMS Alpha encounters an internal problem that hinders system performance without causing a system failure. By allowing you to examine the running system, SDA enables you to search for the solution without disturbing the operating system. For example, you may be able to use SDA to examine the stack and memory of a process that is stalled in a scheduler state, such as a miscellaneous wait (MWAIT) or a suspended (SUSP) state.

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

```
$ ANALYZE/SYSTEM
```

SDA attempts to load SDA$READ_DIR:SYS$BASE_IMAGE.EXE and SDA$READ_DIR:REQSYSDEF.STB. It then executes the contents of any existing SDA initialization file, as it does when invoked to analyze a crash dump (see Sections 2.4 and 2.5, respectively). SDA subsequently displays its identification message and prompt, as follows:

```
OpenVMS Alpha System Analyzer

SDA>
```

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

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

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

  Caution:

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

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

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

- The SHOW CRASH command, although valid, does not display the contents of any of the processor's set of hardware registers. Also, the Time of System Crash information refers to the time at which the ANALYZE/SYSTEM command was given.

4 SDA Context

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

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

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

When you invoke SDA to analyze a crash dump from a multiprocessing system with more than one active CPU, SDA maintains a second dimension of context—its **CPU context**—that allows it to display certain processor-specific information. This information includes the reason for the bugcheck exception, the currently executing process, the current IPL, and the spin locks owned by the processor. When you invoke SDA to analyze a multiprocessor's crash dump, its CPU context defaults to that of the processor that induced the system failure. When you are analyzing a running system, CPU context is not accessible to SDA. Therefore, the SET CPU and SHOW CPU commands are not permitted.

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

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

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

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

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

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

SHOW PROCESS/INDEX=**nn**
SHOW PROCESS/SYSTEM

5 SDA Command Format

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

5.1 General Command Format

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

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

The **command-name** is an SDA command. Each command tells the utility to perform a function. Commands can consist of one or more words, and can be abbreviated to the number of characters that make the command unique. For example, SH stands for SHOW, 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 you supply part of a file specification as a parameter, SDA assumes default values for the omitted portions of the specification. The default device is SYS$DISK, the device specified in your most recent SET DEFAULT command. The default directory is the directory specified in the most recent SET DEFAULT command. See the *OpenVMS DCL Dictionary* for a description of the DCL command SET DEFAULT.

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

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

5.2 Expressions

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

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

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

5.2.1 Radix Operators

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

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

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

5.2.2 Arithmetic and Logical Operators

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

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

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

Table SDA–3 SDA Operators

Operator	Action
Unary Operators	
#	Performs a logical NOT of the expression.
+	Makes the value of the expression positive.
–	Makes the value of the expression negative.
@	Evaluates the following expression as an address, then uses the contents of that address as value.
^Q	When used with the unary operator @, it specifies the size of field to be used as an address is a quadword [1].
^L	When used with the unary operator @, it specifies the size of field to be used as an address is a longword[2].
^W	When used with the unary operator @, it specifies the size of field to be used as an address is a word[3].
^B	When used with the unary operator @, it specifies the size of field to be used as an address is a byte[4].

[1]The command SET FETCH QUADWORD provides the same effect on all subsequent uses of unary operator @ as if ^Q were added each time. That is, SET FETCH is making it the default. For an example of the use of ^Q, see the SET FETCH command.

[2]The command SET FETCH LONGWORD provides the same effect on all subsequent uses of unary operator @ as if ^L were added each time. That is, SET FETCH is making it the default. For an example of the use of ^L, see the SET FETCH command.

[3]The command SET FETCH WORD provides the same effect on all subsequent uses of unary operator @ as if ^W were added each time. That is, SET FETCH is making it the default. For an example of the use of ^W, see the SET FETCH command.

[4]The command SET FETCH BYTE provides the same effect on all subsequent uses of unary operator @ as if ^B were added each time. That is, SET FETCH is making it the default. For an example of the use of ^B, see the SET FETCH command.

(continued on next page)

Table SDA–3 (Cont.) SDA Operators

Operator	Action
Unary Operators	
^P	When used with the unary operator @, it specifies a physical address[5].
^V	When used with the unary operator @, it specifies a virtual address[6].
G	Adds FFFFFFFF 80000000_{16} to the value of the expression[7].
H	Adds $7FFE0000_{16}$ to the value of the expression[8].

I	Fills the leading digits of the following hexadecimal number with hex value of F. For example:
	`SDA> eval i80000000`
	`Hex = FFFFFFFF.80000000 Decimal = -2147483648 G`
	`SYS$PUBLIC_VECTORS_NPRO`

Binary Operators	
+	Addition
–	Subtraction
*	Multiplication
&	Logical AND
|	Logical OR
\	Logical XOR
/	Division[9]
@	Arithmetic shifting

"."	Catenates two 32-bit values into a 64-bit value. For example:
	`SDA> eval fe.50000`
	`Hex = 000000FE00050000  Decimal = 1090922020864`

[5]The command SET FETCH PHYSICAL provides the same effect on all subsequent uses of unary operator @ as if ^P were added each time. That is, SET FETCH is making it the default. For an example of the use of ^P, see the SET FETCH command.

[6]The command SET FETCH VIRTUAL provides the same effect on all subsequent uses of unary operator @ as if ^V were added each time. That is, SET FETCH is making it the default. For an example of the use of ^V, see the SET FETCH command.

[7]The unary operator G corresponds to the first virtual address in system space. For example, the expression GD40 can be used to represent the address FFFFFFFF $80000D40_{16}$.

[8]The unary operator H corresponds to a convenient base address in P1 space ($7FFE0000_{16}$). You can therefore refer to an address such as $7FFE2A64_{16}$ as H2A64.

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

5.2.3 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.

5.2.4 Symbols

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

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

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

```
SDA> EXAMINE/INSTRUCTION procedure descriptor
```

For example, enter the following command:

```
SDA> EXAMINE/INSTRUCTION SCH$QAST
```

SDA displays the following information:

```
SCH$QAST_C:     SUBQ    SP,#X40,SP
```

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

```
SDA> EXAMINE SCH$QAST
```

SDA displays the following information:

```
SCH$QAST:  0000002C.00003009   ".0..,..."
```

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

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

Sources for SDA Symbols

SDA can get its information from the following places:

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

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

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

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

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

Table SDA–4 Modules Containing Global Symbols Used by SDA

File	Contents
DCLDEF.STB	Symbols for the DCL interpreter
DECDTMDEF.STB	Symbols for transaction processing
IMGDEF.STB	Symbols for the image activator
IODEF.STB	I/O database structure symbols
NETDEF.STB	Symbols for DECnet data structures
REQSYSDEF.STB	Required symbols for SDA
RMSDEF.STB	Symbols that define RMS internal and user data structures and RMS$_*xxx* completion codes
SCSDEF.STB	Symbols that define data structures for system communications services
SYSDEF.STB	Symbols that define system data structures, including the I/O database

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

Table SDA–5 SDA Symbols Defined on Initialization

ASN	Address space number
AST	Both the asynchronous system trap status and enable registers: AST<3:0> = AST enable; AST<7:4> = AST status
ESP	Executive stack pointer
FEN	Floating-point enable
FP	Frame pointer (R29)
FP0-FP30	Floating-point registers 0-30
FPCR	Floating-point control register
G	FFFFFFFF.80000000_{16}, the base address of system space
H	00000000.$7FFE0000_{16}$, a base address in P1 space
I	FFFFFFFF.$FFFFFFFF_{16}$ fills the leading digits of a hexadecimal number with the value of F
KSP	Kernel stack pointer
PC	Program counter
PS	Processor status
PTBR	Page table base register

(continued on next page)

Table SDA–5 (Cont.) SDA Symbols Defined on Initialization

R0 through R29	Integer registers
SP	Current stack pointer of a process
SSP	Supervisor stack pointer
USP	User stack pointer

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

Table SDA–6 SDA Symbols Defined by SET CPU Command

IPL	Interrupt priority level register
PCBB	Process context block base register
PRBR	Processor base register (CPU database address)
SCBB	System control block base register
SISR	Software interrupt status register

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

Table SDA–7 SDA Symbols Defined by SET PROCESS Command

ARB	Address of access rights block
JIB	Address of job information block
KTB	Address of the kernel thread block
ORB	Address of object rights block
PCB	Address of process control block
PHD	Address of process header

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

SDA Symbol Initialization

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

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

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

Use of SDA Symbols

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

When the code for one of these commands needs a symbol for an address, it calls the SDA symbolize routine. The symbolize routine tries to find the symbol in the symbol table whose address is closest to, but not greater than the requested address. This means, for any given address, the routine may return a symbol of the form symbol_name+offset. If, however, the offset is greater than $0FFF_{16}$, it fails to find a symbol for the address.

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

executive_image_name+offset
executive_image_name_image_section+offset

The first form is for **nonsliced images**. The offset is the same as the image offset as defined in the map file.

The second form is for a **sliced executive image**. The image sections are not in adjacent locations in memory, so the image section name is needed to find where this address is within the map file. You can also use the MAP command on the address to get the image offset as defined in the map file.

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

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

Symbols and Address Resolution

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

- The SHOW EXECUTIVE command
- The symbolization of addresses, described in the previous section
- The READ command
- The SHOW PROCESS command with the /IMAGES qualifier
- The MAP command

The OpenVMS Alpha executive consists of two base images, SYS$BASE_IMAGE.EXE and SYS$PUBLIC_VECTORS.EXE, and a number of other separately loadable images. Some of these images are loaded on all systems, while others support features unique to particular system configurations. Executive images are mapped into system space during system initialization.

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

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

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

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

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

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

6 Investigating System Failures

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

For a complete description of the commands discussed in the sections that follow, refer to the last part of this document, where all the SDA commands are discussed in alphabetical order.

6.1 General Procedure for Analyzing System Failures

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

To resolve the problem, you must find the reason for the bugcheck. Many failures are caused by errors in user-written device drivers or other privileged code not supplied by 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 Services. To diagnose an error in code supplied by Digital, you need listings of that code, which are available from Digital.

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

However, some bugchecks are caused by unexpected exceptions. In such cases, the address of the instruction that *caused* the exception is more informative than the address of the instruction that signaled the bugcheck. The address of the instruction that caused the exception is located on the stack. You can obtain this address by using the SHOW STACK command to display the contents of the stack or by using the CLUE CRASH command to display the system state at time of exception. See Section 6.2 for information on how to proceed for several types of bugchecks.

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

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

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

6.2 Fatal Bugcheck Conditions

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

6.2.1 Fatal Exceptions

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

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

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

FATALEXCPT, Fatal executive or kernel mode exception

INVEXCEPTN, Exception while above ASTDEL

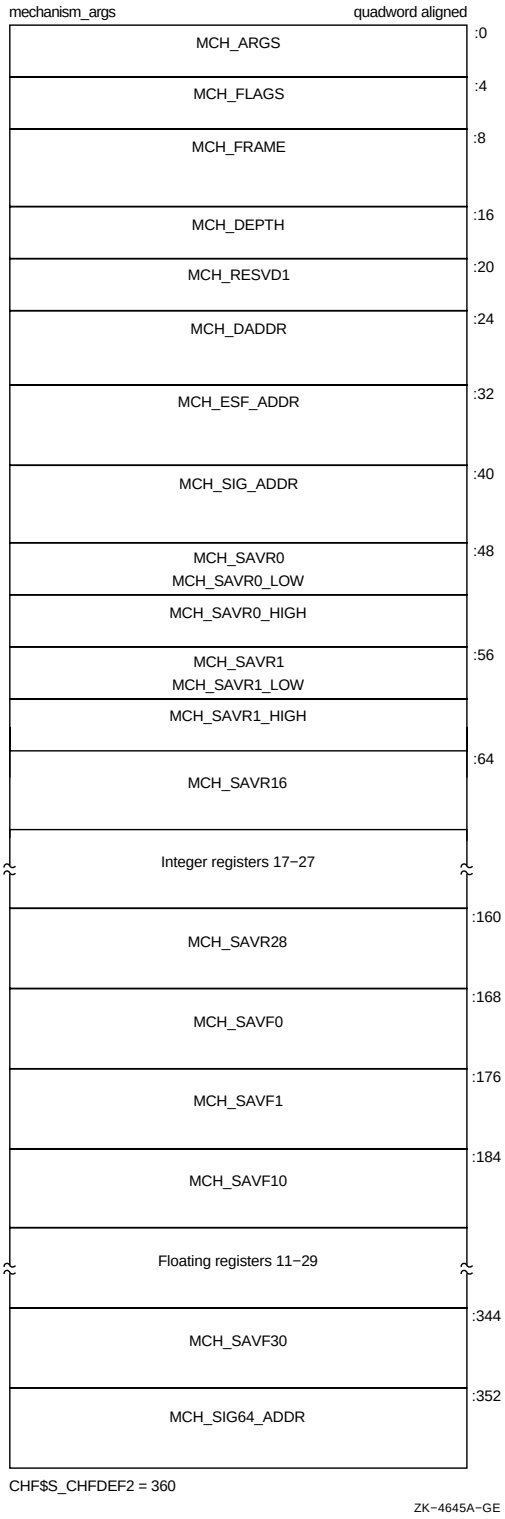
SSRVEXCEPT, Unexpected system service exception

UNXSIGNAL, Unexpected signal name in ACP

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

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

Figure SDA–1 Mechanism Array

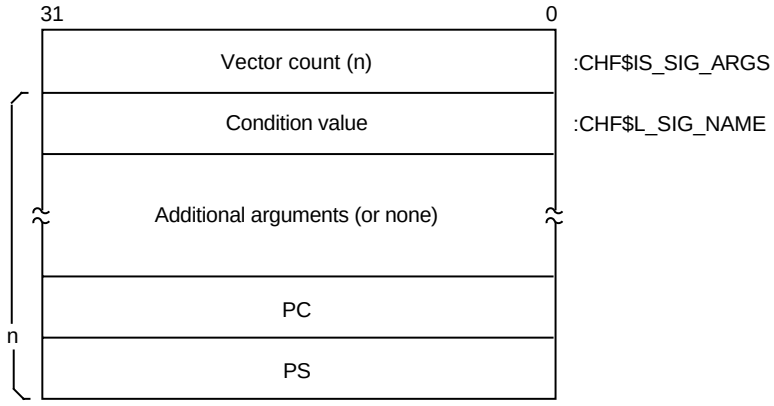


Symbolic offsets into the mechanism array are defined as follows. The SDA SHOW STACK command identifies the elements of the mechanism array on the stack using these symbols.

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

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

Figure SDA–2 Signal Array

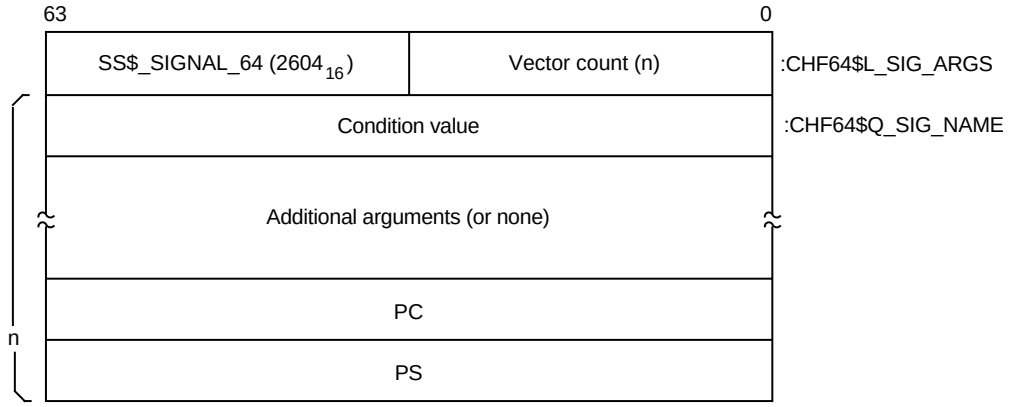


For access violations, the signal array is set up as follows:

Value	Meaning
Vector list length	Number of longwords that follow. For access violations, this value is always 5.
Condition value	Exception code. The value $0C_{16}$ represents an access violation. You can identify the exception code by using the SDA command EVALUATE/CONDITION_VALUE or SHOW CRASH.
Additional arguments	These can include a reason mask and a virtual address. In the longword mask if bit 0 of the longword is set, the failing instruction (at the PC saved below) caused a length violation. If bit 1 is set, it referred to a location whose page table entry is in a "no access" page. Bit 2 indicates the type of access used by the failing instruction: it is set for write and modify operations and clear for read operations. The virtual address represents the low-order 32 bits of the virtual address that the failing instruction tried to reference.
PC	PC whose execution resulted in the exception.
PS	PS at the time of the exception.

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

Figure SDA–3 64-Bit Signal Array

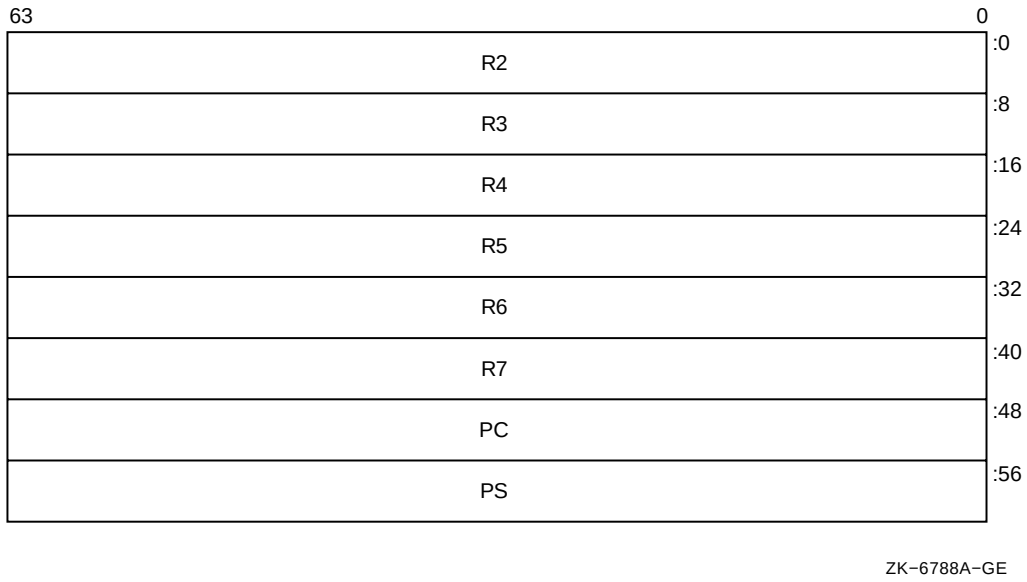


For access violations, the 64-bit signal array is set up as follows:

Value	Meaning
Vector list length	Number of quadwords that follow. For access violations, this value is always 5.
Condition value	Exception code. The value $0C_{16}$ represents an access violation. You can identify the exception code by using the SDA command EVALUATE/CONDITION_VALUE or SHOW CRASH.
Additional arguments	These can include a reason mask and a virtual address. In the quadword mask if bit 0 of the quadword is set, the failing instruction (at the PC saved below) caused a length violation. If bit 1 is set, it referred to a location whose page table entry is in a "no access" page. Bit 2 indicates the type of access used by the failing instruction: it is set for write and modify operations and clear for read operations.
PC	PC whose execution resulted in the exception.
PS	PS at the time of the exception.

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

Figure SDA–4 Exception Stack Frame



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

Table SDA–8 Exception Stack Frame Values

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

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

If OpenVMS Alpha encounters a fatal exception, you can find the code that signaled it by examining the PC in the signal array. Use the SHOW CRASH or CLUE CRASH command to display the PC and the instruction stream around the PC to locate the exception.

The following display shows the SDA output in response to SHOW CRASH and SHOW STACK commands for an SSRVEXCEPT bugcheck. It illustrates the mechanism array, signal arrays, and exception stack frame previously described.

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

Dump taken on 30-AUG-1996 13:13:46.83
SSRVEXCEPT, Unexpected system service exception

SDA>  SHOW CRASH
Time of system crash: 30-AUG-1996 13:13:46.83

Version of system: OpenVMS (TM) Alpha Operating System, Version X6AF-FT2

System Version Major ID/Minor ID: 3/0

System type: DEC 3000 Model 400

Crash CPU ID/Primary CPU ID:  00/00

Bitmask of CPUs active/available:  00000001/00000001

CPU bugcheck codes:
        CPU 00 -- SSRVEXCEPT, Unexpected system service exception

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

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

   PS =>
         MBZ SPAL      MBZ    IPL VMM MBZ CURMOD INT PRVMOD
         0   00   00000000000 00  0   0   KERN   0   USER

Signal Array
------------
        Length = 00000005
        Type   = 0000000C
        Arg    = 00000000.00010000
        Arg    = 00000000.00000000
        Arg    = 00000000.00030078
        Arg    = 00000000.00000003
%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=0000000000000000,
   PC=0000000000030078, PS=00000003

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

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

CPU 00 reason for Bugcheck: SSRVEXCEPT, Unexpected system service exception

Process currently executing on this CPU: SYSTEM

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

Current IPL: 0  (decimal)

CPU database address:  80D0E000

CPUs Capabilities:    PRIMARY,QUORUM,RUN

General registers:

R0   = 00000000.00000000  R1   = 00000000.7FFA1EB8  R2   = FFFFFFFF.80D0E6C0
R3   = FFFFFFFF.80C63460  R4   = FFFFFFFF.80D12740  R5   = 00000000.000000C8
R6   = 00000000.00030038  R7   = 00000000.7FFA1FC0  R8   = 00000000.7FFAC208
R9   = 00000000.7FFAC410  R10  = 00000000.7FFAD238  R11  = 00000000.7FFCE3E0
R12  = 00000000.00000000  R13  = FFFFFFFF.80C6EB60  R14  = 00000000.00000000
R15  = 00000000.009A79FD  R16  = 00000000.000003C4  R17  = 00000000.7FFA1D40
R18  = FFFFFFFF.80C05C38  R19  = 00000000.00000000  R20  = 00000000.7FFA1F50
R21  = 00000000.00000000  R22  = 00000000.00000001  R23  = 00000000.7FFF03C8
R24  = 00000000.7FFF0040  AI   = 00000000.00000003  RA   = FFFFFFFF.82A21080
PV   = FFFFFFFF.829CF010  R28  = FFFFFFFF.8004B6DC  FP   = 00000000.7FFA1CA0
PC   = FFFFFFFF.82A210B4  PS   = 18000000.00000000


Processor Internal Registers:

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

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

        KSP    = 00000000.7FFA1C98
        ESP    = 00000000.7FFA6000
        SSP    = 00000000.7FFAC100
        USP    = 00000000.7AFFBAD0

                No spinlocks currently owned by CPU 00
```

```
SDA> SHOW STACK
Current Operating Stack (KERNEL):
                        00000000.7FFA1C78    18000000.00000000
                        00000000.7FFA1C80    00000000.7FFA1CA0
                        00000000.7FFA1C88    00000000.00000000
                        00000000.7FFA1C90    00000000.7FFA1D40
                  SP => 00000000.7FFA1C98    00000000.00000000
                        00000000.7FFA1CA0    FFFFFFFF.829CF010  EXE$EXCPTN
                        00000000.7FFA1CA8    FFFFFFFF.82A2059C  EXCEPTION_MON_PRO+0259C
                        00000000.7FFA1CB0    00000000.00000000
                        00000000.7FFA1CB8    00000000.7FFA1CD0
                        00000000.7FFA1CC0    FFFFFFFF.829CEDA8  EXE$SET_PAGES_READ_ONLY+00948
                        00000000.7FFA1CC8    00000000.00000000
                        00000000.7FFA1CD0    FFFFFFFF.829CEDA8  EXE$SET_PAGES_READ_ONLY+00948
                        00000000.7FFA1CD8    00000000.00000000
                        00000000.7FFA1CE0    FFFFFFFF.82A1E930  EXE$CONTSIGNAL_C+001D0
                        00000000.7FFA1CE8    00000000.7FFA1F40
                        00000000.7FFA1CF0    FFFFFFFF.80C63780  EXE$ACVIOLAT
                        00000000.7FFA1CF8    00000000.7FFA1EB8
                        00000000.7FFA1D00    00000000.7FFA1D40
                        00000000.7FFA1D08    00000000.7FFA1F00
                        00000000.7FFA1D10    00000000.7FFA1F40
                        00000000.7FFA1D18    00000000.00000000
                        00000000.7FFA1D20    00000000.00000000
                        00000000.7FFA1D28    00000000.00020000  SYS$K_VERSION_04
                        00000000.7FFA1D30    00000005.00000250  BUG$_NETRCVPKT
                        00000000.7FFA1D38    829CE050.000008F8  BUG$_SEQ_NUM_OVF
CHF$IS_MCH_ARGS         00000000.7FFA1D40    00000000.0000002C
CHF$PH_MCH_FRAME        00000000.7FFA1D48    00000000.7AFFBAD0
CHF$IS_MCH_DEPTH        00000000.7FFA1D50    FFFFFFFF.FFFFFFFD
CHF$PH_MCH_DADDR        00000000.7FFA1D58    00000000.00000000
CHF$PH_MCH_ESF_ADDR     00000000.7FFA1D60    00000000.7FFA1F00
CHF$PH_MCH_SIG_ADDR     00000000.7FFA1D68    00000000.7FFA1EB8
CHF$IH_MCH_SAVR0        00000000.7FFA1D70    00000000.00020000  SYS$K_VERSION_04
CHF$IH_MCH_SAVR1        00000000.7FFA1D78    00000000.00000000
CHF$IH_MCH_SAVR16       00000000.7FFA1D80    00000000.00020004  UCB$M_LCL_VALID+00004
CHF$IH_MCH_SAVR17       00000000.7FFA1D88    00000000.00010050  SYS$K_VERSION_16+00010
CHF$IH_MCH_SAVR18       00000000.7FFA1D90    FFFFFFFF.FFFFFFFF
CHF$IH_MCH_SAVR19       00000000.7FFA1D98    00000000.00000000
CHF$IH_MCH_SAVR20       00000000.7FFA1DA0    00000000.7FFA1F50
CHF$IH_MCH_SAVR21       00000000.7FFA1DA8    00000000.00000000
CHF$IH_MCH_SAVR22       00000000.7FFA1DB0    00000000.00010050  SYS$K_VERSION_16+00010
CHF$IH_MCH_SAVR23       00000000.7FFA1DB8    00000000.00000000
CHF$IH_MCH_SAVR24       00000000.7FFA1DC0    00000000.00010051  SYS$K_VERSION_16+00011
CHF$IH_MCH_SAVR25       00000000.7FFA1DC8    00000000.00000000
CHF$IH_MCH_SAVR26       00000000.7FFA1DD0    FFFFFFFF.8010ACA4  AMAC$EMUL_CALL_NATIVE_C+000A4
CHF$IH_MCH_SAVR27       00000000.7FFA1DD8    00000000.00010050  SYS$K_VERSION_16+00010
CHF$IH_MCH_SAVR28       00000000.7FFA1DE0    00000000.00000000
                        00000000.7FFA1DE8    00000000.00000000
                        00000000.7FFA1DF0    00000000.00000000
                        00000000.7FFA1DF8    00000000.00000000
                        00000000.7FFA1E00    00000000.00000000
                        00000000.7FFA1E08    00000000.00000000
                        00000000.7FFA1E10    00000000.00000000
                        00000000.7FFA1E18    00000000.00000000
                        00000000.7FFA1E20    00000000.00000000
                        00000000.7FFA1E28    00000000.00000000
                        00000000.7FFA1E30    00000000.00000000
                        00000000.7FFA1E38    00000000.00000000
                        00000000.7FFA1E40    00000000.00000000
                        00000000.7FFA1E48    00000000.00000000
                        00000000.7FFA1E50    00000000.00000000
                        00000000.7FFA1E58    00000000.00000000
                        00000000.7FFA1E60    00000000.00000000
                        00000000.7FFA1E68    00000000.00000000
```

```
                        00000000.7FFA1E70    00000000.00000000
                        00000000.7FFA1E78    00000000.00000000
                        00000000.7FFA1E80    00000000.00000000
                        00000000.7FFA1E88    00000000.00000000
                        00000000.7FFA1E90    00000000.00000000
                        00000000.7FFA1E98    00000000.00000000
CHF$PH_MCH_SIG64_ADDR   00000000.7FFA1EA0    00000000.7FFA1ED0
                        00000000.7FFA1EA8    00000000.00000000
                        00000000.7FFA1EB0    00000000.7FFA1F50
                        00000000.7FFA1EB8    0000000C.00000005
                        00000000.7FFA1EC0    00000000.00010000  SYS$K_VERSION_07
                        00000000.7FFA1EC8    00000003.00030078  SYS$K_VERSION_01+00078
CHF$L_SIG_ARGS          00000000.7FFA1ED0    00002604.00000005  UCB$M_TEMPLATE+00604
CHF$L_SIG_ARG1          00000000.7FFA1ED8    00000000.0000000C
                        00000000.7FFA1EE0    00000000.00010000  SYS$K_VERSION_07
                        00000000.7FFA1EE8    00000000.00000000
                        00000000.7FFA1EF0    00000000.00030078  SYS$K_VERSION_01+00078
                        00000000.7FFA1EF8    00000000.00000003
INTSTK$Q_R2             00000000.7FFA1F00    00000000.00000003
INTSTK$Q_R3             00000000.7FFA1F08    FFFFFFFF.80C63460  EXCEPTION_MON_NPRW+06A60
INTSTK$Q_R4             00000000.7FFA1F10    FFFFFFFF.80D12740  PCB
INTSTK$Q_R5             00000000.7FFA1F18    00000000.000000C8
INTSTK$Q_R6             00000000.7FFA1F20    00000000.00030038  SYS$K_VERSION_01+00038
INTSTK$Q_R7             00000000.7FFA1F28    00000000.7FFA1FC0
INTSTK$Q_PC             00000000.7FFA1F30    00000000.00030078  SYS$K_VERSION_01+00078
INTSTK$Q_PS             00000000.7FFA1F38    00000000.00000003
Prev SP (7FFA1F40) ==>  00000000.7FFA1F40    00000000.00010050  SYS$K_VERSION_16+00010
                        00000000.7FFA1F48    00000000.00010000  SYS$K_VERSION_07
                        00000000.7FFA1F50    FFFFFFFF.8010ACA4  AMAC$EMUL_CALL_NATIVE_C+000A4
                        00000000.7FFA1F58    00000000.7FFA1F70
                        00000000.7FFA1F60    00000000.00000001
                        00000000.7FFA1F68    FFFFFFFF.800EE81C  RM_STD$DIRCACHE_BLKAST_C+005AC
                        00000000.7FFA1F70    FFFFFFFF.80C6EBA0  SCH$CHSEP+001E0
                        00000000.7FFA1F78    00000000.829CEDE8  EXE$SIGTORET
                        00000000.7FFA1F80    00010050.00000002  SYS$K_VERSION_16+00010
                        00000000.7FFA1F88    00000000.00020000  SYS$K_VERSION_04
                        00000000.7FFA1F90    00000000.00030000  SYS$K_VERSION_01
                        00000000.7FFA1F98    FFFFFFFF.800A4D64  EXCEPTION_MON_NPRO+00D64
                        00000000.7FFA1FA0    00000000.00000003
                        00000000.7FFA1FA8    FFFFFFFF.80D12740  PCB
                        00000000.7FFA1FB0    00000000.00010000  SYS$K_VERSION_07
                        00000000.7FFA1FB8    00000000.7AFFBAD0
                        00000000.7FFA1FC0    00000000.7FFCF880  MMG$IMGHDRBUF+00080
                        00000000.7FFA1FC8    00000000.7B0E9851
                        00000000.7FFA1FD0    00000000.7FFCF818  MMG$IMGHDRBUF+00018
                        00000000.7FFA1FD8    00000000.7FFCF938  MMG$IMGHDRBUF+00138
                        00000000.7FFA1FE0    00000000.7FFAC9F0
                        00000000.7FFA1FE8    00000000.7FFAC9F0
                        00000000.7FFA1FF0    FFFFFFFF.80000140  SYS$PUBLIC_VECTORS_NPRO+00140
                        00000000.7FFA1FF8    00000000.0000001B

   .
   .
   .
```

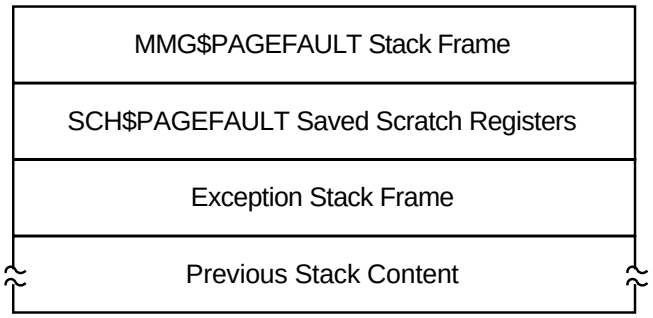
6.2.2 Illegal Page Faults

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

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

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

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



The stack contents are as follows:

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

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

7 Inducing a System Failure

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

If the system has suspended all noticeable activity and is hung, see the examples of causing system failures in Section 7.2.

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

7.1 Meeting Crash Dump Requirements

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

- You must not halt the system until the console dump messages have been printed in their entirety and the memory contents have been written to the crash dump file. Be sure to allow sufficient time for these events to take place or make sure that all disk activity has stopped before using the console to halt the system.
- There must be a crash dump file in SYS$SPECIFIC:[SYSEXE]: named either SYSDUMP.DMP or PAGEFILE.SYS.

  This dump file must be either large enough to hold the entire contents of memory (as discussed in Section 1.1.1) or, if the DUMPSTYLE system parameter is set, large enough to accommodate a subset or compressed dump (also discussed in Section 1.1.1).

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

7.2 Procedure for Causing a System Failure

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

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

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

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

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

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

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

```
>>> DEPOSIT PC FFFFFFFFFFFFFF00
>>> DEPOSIT PS 1F00
>>> CONTINUE
```

For more information, refer to the hardware manuals that accompanied your computer.

SDA Usage Summary

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

Format

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

Command Parameter

filespec

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

Description

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

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

```
$ ANALYZE/CRASH_DUMP filespec
```

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

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

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

```
$ANALYZE/SYSTEM
```

To examine a running system, your process must have change-mode-to-kernel (CMKRNL) privilege. You cannot specify **filespec** when using the /SYSTEM qualifier.

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

```
SDA> SET OUTPUT SYS$OUTPUT
```

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

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

SDA Qualifiers

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

/CRASH_DUMP
/OVERRIDE
/RELEASE
/SYMBOL
/SYSTEM

/CRASH_DUMP

Invokes SDA to analyze the specified dump file.

Format

/CRASH_DUMP filespec

Parameter

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

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

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

Description

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

Examples

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

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

2.
```
$ ANALYZE/CRASH SYS$SYSTEM:PAGEFILE.SYS
```

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

/OVERRIDE

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

Format

/CRASH_DUMP/OVERRIDE filespec

Parameter

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

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

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

Description

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

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

Commands that cannot be used are as follows:

- Commands that access memory addresses within the dump file such as EXAMINE and SHOW SUMMARY
- You cannot specify the /RELEASE qualifier when you include the /OVERRIDE qualifier in the ANALYZE/CRASH_DUMP command.

Examples

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

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

/RELEASE

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

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

Format

/RELEASE filespec

Parameter

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

Description

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

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

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

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

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

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

Example

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

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

/SYMBOL

Specifies an alternate system symbol table for SDA to use.

Format

/SYMBOL =system-symbol-table

Parameter

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

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

Description

The /SYMBOL qualifier allows you to specify a system symbol table to load into the SDA symbol table. You can use the /SYMBOL qualifier whether you are analyzing a system dump or a running system.

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

Example

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

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

/SYSTEM

Invokes SDA to analyze a running system.

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

Format

/SYSTEM

Parameters

None.

Description

See Section 3 to use SDA to analyze a running system.

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

Example

```
$ ANALYZE/SYSTEM
```

This command invokes SDA to analyze the running system.

SDA Commands

The following SDA commands, which are described in this section, can be used to analyze a system dump or a running system. SDA CLUE extension commands, which can summarize information provided by certain SDA commands and provide additional detail for some SDA commands, are described in the following section.

@ (Execute Command)
ATTACH
COPY
DEFINE
DEFINE/KEY
EVALUATE
EXAMINE
EXIT
FORMAT
HELP
MAP
MODIFY DUMP
READ
REPEAT
SEARCH
SET CPU
SET ERASE_SCREEN
SET FETCH
SET LOG
SET OUTPUT
SET PROCESS
SET RMS
SET SIGN_EXTEND
SHOW ADDRESS
SHOW BUGCHECK
SHOW CALL_FRAME
SHOW CLUSTER
SHOW CONNECTIONS
SHOW CPU
SHOW CRASH
SHOW DEVICE
SHOW DUMP
SHOW EXECUTIVE
SHOW GLOBAL_SECTION_TABLE
SHOW GSD
SHOW HEADER
SHOW LAN
SHOW LOCK
SHOW MACHINE_CHECK
SHOW PAGE_TABLE
SHOW PFN_DATA
SHOW POOL
SHOW PORTS
SHOW PROCESS
SHOW RESOURCE
SHOW RMD
SHOW RMS

SHOW RSPID
SHOW SPINLOCKS
SHOW STACK
SHOW SUMMARY
SHOW SYMBOL
SHOW WORKING_SET_LIST
SPAWN
VALIDATE PFN_LIST
VALIDATE QUEUE

@ (Execute Command)

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

Format

@filespec

Parameter

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

Example

```
SDA> @USUAL
```

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

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

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

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

Command procedures cannot be nested.

ATTACH

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

Format

ATTACH [/PARENT] process-name

Parameter

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

Qualifier

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

Examples

1. `SDA> ATTACH/PARENT`

   This ATTACH command attaches the terminal to the parent process of the current process.

2. `SDA> ATTACH DUMPER`

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

COPY

Copies the contents of the dump file to another file.

Format

COPY [/qualifier...] output-filespec

Parameter

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

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

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

Qualifiers

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

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

Description

Each time the system fails, it copies the contents of memory and the hardware context of the current process (as directed by the DUMPSTYLE parameter) into the file SYS$SYSTEM:SYSDUMP.DMP (or the page file), overwriting its contents. Each time the system is shut down normally, it overwrites the dump file with error log messages that have not yet been written to the error log file. If you do not save this crash dump elsewhere, it will be overwritten the next time that the system fails or is shut down.

The COPY command allows you to preserve a crash dump by copying its contents to another file. It is generally useful to invoke SDA during system initialization (from within SYS$MANAGER:SYSTARTUP_VMS.COM) to execute the COPY command. This ensures that a copy of the dump file is made only after the system has failed.

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

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

may be lost. Perform subsequent analysis upon the copy of the dump created by the COPY command.

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

Example

```
SDA> COPY 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 [/qualifier...] symbol-name [=] expression

Parameters

symbol-name
Name, containing from 1 to 31 alphanumeric characters, that identifies the symbol. See Section 5.2.4 for a description of SDA symbol syntax and a list of default symbols.

expression
Definition of the symbol's value. See Section 5.2 for a discussion of the components of SDA expressions.

Qualifier

/PD
Defines a symbol as a procedure descriptor (PD). It also defines the routine address symbol corresponding to the defined symbol (the routine address symbol has the same name as the defined symbol, only with _C appended to the symbol name). See Section 5.2.4 for more information about symbols.

Description

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

Examples

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

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

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

The symbol NEXT defines the address contained in the program counter, so that the symbol can be used in an EXAMINE/INSTRUCTION command.

3.
```
SDA>  DEFINE VEC SCH$GL_PCBVEC
SDA>  EXAMINE VEC
SCH$GL_PCBVEC:  00000000.8060F2CC   "Ìò‘....."
SDA>
```

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

4.
```
SDA>  DEFINE/PD VEC SCH$QAST
SDA>  EXAMINE VEC
SCH$QAST:  0000002C.00003008   ".0..,..."
SDA>  EXAMINE VEC_C
SCH$QAST_C:  B75E0008.43C8153E   ">.ÈC..^·"
SDA>
```

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

DEFINE/KEY

Associates an SDA command with a terminal key.

Format

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

Parameters

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

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

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

Qualifiers

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

/PD
Defines a symbol as a procedure descriptor (PD). Also defines the routine address symbol corresponding to the defined symbol (the routine address symbol has the same name as the defined symbol, only with _C appended to the symbol name.)

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

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

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

Description

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

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

Examples

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

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

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

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

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

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

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

EVALUATE

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

Format

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

Parameter

expression
SDA expression to be evaluated. Section 5.2 describes the components of SDA expressions.

Qualifiers

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

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

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

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

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

Description

If the expression is equal to the value of a symbol in the SDA symbol table, that symbol is displayed. If no symbol with this value is known, the next lower valued symbol is displayed with an appropriate offset unless the offset is extremely large. The DEFINE command adds symbols to the SDA symbol table but does not display the results of the computation. EVALUATE displays the result of the computation but does not add symbols to the SDA symbol table.

If no qualifier is specified, the EVALUATE command interprets and displays the expression as hexadecimal and decimal values.

Examples

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

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

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

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

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

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

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

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

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

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

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

6.
```
SDA> EVALUATE/PS  0B03
      MBZ SPAL      MBZ    IPL VMM MBZ CURMOD INT PRVMOD
      0   00   00000000000 0B  0   0   KERN   0   USER
```

   SDA interprets the entered value 0B03 as though it were a processor status (PS) and displays the resulting field values.

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

```
3 3 2  2             2   1   1 1
1 0 9  7             0   8   6 5               7 6           0
+-+-+--+-------------+-+-+---+-+---------------+-+-----------+-+
|0|0|00|     005E    |0|X| 02|1|      FF       |X|    37     |0|
+-+-+--+-------------+-+-+---+-+---------------+-+-----------+-+
|                            00000000                         |
+-------------------------------------------------------------+
Global PTE:  Owner = S, Read Prot = KESU, Write Prot = KESU, CPY = 0
                    GPT Index  = 00000000
```

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

8.
```
SDA>EVALUATE/TIME 009A9A4C.843DBA9F
10-OCT-1996 15:59:44.02
```

   This example shows the use of the EVALUATE/TIME command.

EXAMINE

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

Format

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

Parameter

location

Location in memory to be examined. A location can be represented by any valid SDA expression. (See Section 5.2 for additional information about expressions.) To examine a range of locations, the following syntax is used:

m:n Range of locations to be examined, from *m* to *n*

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

The default location that SDA uses is initially 0 in the program region (P0) of the process that was executing at the time the system failed (if you are examining a crash dump) or your process (if you are examining the running system). Subsequent uses of the EXAMINE command with no parameter specified increase the last address examined by 8. Use of the /INSTRUCTION qualifier increases the default address by 4. To examine memory locations of other processes, you must use the SET PROCESS command.

Qualifiers

/ALL

Examines all the locations in the program, and control regions and parts of the writable system region, displaying the contents of memory in hexadecimal longwords. Do not specify parameters when you use this qualifier.

/CONDITION_VALUE

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

/INSTRUCTION

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

/NOPD

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

/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.

/PD
Causes the EXAMINE command to treat the location specified in the EXAMINE command as a procedure descriptor (PD). PD can also be used to qualify symbols.

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

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

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

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

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

Description

The following sections describe how to use the EXAMINE command.

Examining Locations

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

```
SDA> EXAMINE G6605C0
806605C0:  64646464.64646464  "dddddddd"
```

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

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

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

```
SDA> EXAMINE G40:G200
```

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

```
SDA> EXAMINE G400;16
```

When used to display the contents of a range of locations, the EXAMINE command displays six columns of information:

- Each of the first four columns represents a longword of memory, the contents of which are displayed in hexadecimal format.
- The fifth column lists the ASCII value of each byte in each longword displayed in the previous four columns.
- The sixth column contains the address of the first, or rightmost, longword in each line. This address is also the address of the first, or leftmost, character in the ASCII representation of the longwords. Thus, you read the hexadecimal dump display from right to left, and the ASCII display from left to right.

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

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

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

Decoding Locations

You can translate the contents of memory locations into instruction format by using the /INSTRUCTION qualifier. This qualifier causes SDA to display the location in symbolic notation (if possible) and its contents in instruction format. The operands of decoded instructions are also displayed in symbolic notation. The location must be longword assigned.

Examining Memory Regions

You can display an entire region of virtual memory by using one or more of the qualifiers /ALL, /SYSTEM, /P0, and /P1 with the EXAMINE command.

Other Uses

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

Examples

1.
```
SDA> EXAMINE/PS 7FF95E78
          MBZ SPAL      MBZ    IPL VMM MBZ CURMOD INT PRVMOD
          0   00   00000000000 08  0   0   KERN   0   EXEC
```

   This example shows the display produced by the EXAMINE/PS command.

2.
```
SDA>EXAMINE/PTE @^QMMG$GG_L1_BASE



 3 3 2  2             2   1   1 1
 1 0 9  7             0   8   6 5               7 6          0
+-+-+--+-------------+-+-+---+-+---------------+-+----------+-+
|0|1|00|     0000    |0|X| 00|0|      11       |X|    04    |1|
+-+-+--+-------------+-+-+---+-+---------------+-+----------+-+
|                           00000C37                          |
+-------------------------------------------------------------+
Valid PTE: Read Prot = K---, Write Prot = K---
           Owner = K, Fault on = -E--, ASM = 00, Granularity Hint = 00
           CPY = 00  PFN = 00000C37
```

   The EXAMINE/PTE command displays and formats the level 1 page table entry at FFFFFFFF.FF7FC000.

EXIT

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

Format

EXIT

Parameters

None.

Qualifiers

None.

Description

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

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

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

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

FORMAT

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

Format

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

Parameter

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

Qualifiers

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

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

Description

The FORMAT command performs the following actions:

- Characterizes a range of locations as a system data block
- Assigns, if possible, a symbol to each item of data within the block
- Displays all the data within the block

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

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

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

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

```
%SDA-E-NOSYMBOLS, no "block-type" symbols found to format this block
```

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

```
SDA> READ SDA$READ_DIR:SYSDEF.STB
```

If SDA issues the same message again, try reading additional symbols. Table SDA–4 lists additional modules provided by the OpenVMS operating system. Alternatively, you can create your own object modules with the MACRO-32 Compiler for OpenVMS Alpha.

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

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

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

The FORMAT command produces a 3-column display:

- The first column shows the virtual address of each item within the block.
- The second column lists each symbolic name associated with a location within the block.
- The third column shows the contents of each item in hexadecimal format.

Example

```
SDA>READ SDA$READ_DIR:SYSDEF.STB
%SDA-I-READSYM, 913 symbols read from SYS$COMMON:[SYS$LDR]SYSDEF.STB
SDA>FORMAT G41F818
FFFFFFFF.8041F818   UCB$L_FQFL                      8041F818 UCB
                    UCB$L_MB_MSGQFL
                    UCB$L_RQFL
                    UCB$W_MB_SEED
                    UCB$W_UNIT_SEED
FFFFFFFF.8041F81C   UCB$L_FQBL                      8041F818 UCB
                    UCB$L_MB_MSGQBL
                    UCB$L_RQBL
FFFFFFFF.8041F820   UCB$W_SIZE                          0110
FFFFFFFF.8041F822   UCB$B_TYPE                        10
FFFFFFFF.8041F823   UCB$B_FLCK                      2C
FFFFFFFF.8041F824   UCB$L_ASTQFL                    00000000
                    UCB$L_FPC
                    UCB$L_MB_W_AST
                    UCB$T_PARTNER
   .
   .
   .
```

The READ command loads into SDA's symbol table the symbols from SDA$READ_DIR:SYSDEF.STB. The FORMAT command displays the data structure that begins at $G41F818_{16}$, a unit control block (UCB). If a field has more than one symbolic name, all such names are displayed. Thus, the field that starts at $8041F824_{16}$ has four designations: UCBL_ASTQFL, UCBL_FPC, UCB$L_MB_W_AST, and UCB$T_PARTNER.

The contents of each field appear to the right of the symbolic name of the field. Thus, the contents of UCB$L_FQBL are $8041F818_{16}$.

HELP

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

Format

HELP [command-name]

Parameter

command-name
Command for which you need information.

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

Keyword	Function
CPU_CONTEXT	Describes the concept of CPU context as it governs the behavior of SDA.
EXECUTE_COMMAND	Describes the use of @ file to execute SDA commands contained in a file.
EXPRESSIONS	Prints a description of SDA expressions.
INITIALIZATION	Describes the circumstances under which SDA executes an initialization file when first invoked.
OPERATION	Describes how to operate SDA at your terminal and by means of the site-specific startup procedure.
PROCESS_CONTEXT	Describes the concept of process context as it governs the behavior of SDA.
SYMBOLS	Describes the symbols used by SDA.

Qualifiers

None.

Description

The HELP command displays brief descriptions of SDA commands and concepts on the terminal screen (or sends these descriptions to the file designated in a SET OUTPUT command). You can request additional information by specifying the name of a topic in response to the Topic? prompt.

If you do not specify a parameter in the HELP command, it lists those commands and topics for which you can request help, as follows:

```
  Information available:

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

Topic?
```

MAP

Transforms an address into an offset in a particular image.

Format

MAP address

Parameter

address
Address to be identified.

Qualifiers

None.

Description

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

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

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

Examples

1.
```
SDA> MAP G90308
Image                                Base      End       Image Offset
SYS$VM
Nonpaged read only                   80090000  800ABA00  00000308
```

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

```
Psect Name      Module Name       Base     End          Length       Align
----------      -----------       ----     ---          ------       -----

$CODE$                        00000000 00000000 00000000 (     0.) QUAD  3 . . .
                BUGCHECK_CODES  00000000 00000000 00000000 (     0.) QUAD  3

$GLOBAL$                      00000000 00000000 00000000 (     0.) QUAD  3 . . .
                BUGCHECK_CODES  00000000 00000000 00000000 (     0.) QUAD  3

$LINK$                        00000000 00000000 00000000 (     0.) QUAD  3 . . .
                BUGCHECK_CODES  00000000 00000000 00000000 (     0.) QUAD  3

$OWN$                         00000000 00000000 00000000 (     0.) QUAD  3 . . .
                BUGCHECK_CODES  00000000 00000000 00000000 (     0.) QUAD  3

$PLIT$                        00000000 00000000 00000000 (     0.) QUAD  3 . . .
                BUGCHECK_CODES  00000000 00000000 00000000 (     0.) QUAD  3

. LITERAL  .                  00000000 00000000 00000000 (     0.) QUAD  3 . . .
                BUGCHECK_CODES  00000000 00000000 00000000 (     0.) QUAD  3
```

```
. BLANK .                       00000000 00000000 00000000 (     0.) OCTA  4 . . .
                SYS$DOINIT      00000000 00000000 00000000 (     0.) OCTA  4
                EXECUTE_FAULT   00000000 00000000 00000000 (     0.) OCTA  4
                GSD_ROUTINES    00000000 00000000 00000000 (     0.) OCTA  4
                IOLOCK          00000000 00000000 00000000 (     0.) OCTA  4
   .
   .
   .
EXEC$HI_USE_PAGEABLE_CODE       00000000 000045D3 000045D4 ( 17876.) 2 **  5 . . .
                SYSCREDEL       00000000 0000149B 0000149C (  5276.) 2 **  5
                SYSCRMPSC       000014A0 000045D3 00003134 ( 12596.) 2 **  5
EXEC$NONPAGED_CODE              000045E0 0001B8B3 000172D4 ( 94932.) 2 **  5 . . .
                EXECUTE_FAULT   000045E0 0000483B 0000025C (   604.) 2 **  5
                IOLOCK          00004840 000052E7 00000AA8 (  2728.) 2 **  5
                LOCK_SYSTEM_PAGES
   .
   .
   .
```

Specifically, image offset 308 is located within source module SYSCREDEL. Therefore, to locate the corresponding code, you would look in SYSCREDEL for offset 308 in psect EXEC$HI_USE_PAGEABLE_CODE.

2.
```
SDA> MAP G550000
Image                           Base         End       Image Offset
SYS$DKDRIVER                    80548000     80558000  00008000
```

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

3.
```
SDA> MAP G550034
Image                           Base         End       Image Offset
SYS$DUDRIVER
      Nonpaged read/write       80550000     80551400  00008034
```

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

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

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

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

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

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

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

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

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

8.
```
SDA> MAP 7FC26000
Read-Write Data Section                 Base      End       Image Offset
LIBRTL                                  7FC26000  7FC27000  000B0000
```

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

9.
```
SDA> MAP 7FC36000
Shareable Read-Only Data Section        Base      End       Image Offset
LIBRTL                                  7FC36000  7FC3F600  000C0000
```

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

10.
```
SDA> MAP 7FC56000
Demand Zero Data Section                Base      End       Image Offset
LIBRTL                                  7FC56000  7FC57000  000E0000
```

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

MODIFY DUMP

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

Format

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

Parameter

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

Qualifiers

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

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

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

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

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

/WORD
Indicates that a word is to be replaced.

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

/QUADWORD
Indicates that a quadword is to be replaced.

Description

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

Important

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

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

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

```
%SDA-W-DUMPMOD, dump has been modified
```

Example

```
SDA> MODIFY DUMP/BLOCK=10/OFFSET=100/WORD FF
```

This example shows the dump file modified with word value of 0000 at offset 100 in block 00000010 replaced by 00FF.

```
SDA> MODIFY DUMP/BLOCK=10/OFFSET=100/WORD 0/CONFIRM=EE
```

This example shows that the actual word value of 00FF at offset 100 in block 00000010 does not match given value of 00EE. The following message is displayed:

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

```
SDA> MODIFY DUMP/BLOCK=10/OFFSET=100/WORD 0/CONFIRM=FF
```

This example shows the dump file modified with a word value of 00FF at offset 100 in block 00000010 replaced by 0000.

READ

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

Format

READ [/LOG|/NOLOG|/RELOCATE =*expression*|/SYMVA=*expression*]
{/EXECUTIVE [directory spec]|/FORCE filespec
|/IMAGE filespec|filespec}

Parameters

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

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

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

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

Qualifiers

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

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

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

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

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

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

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

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

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

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

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

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

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

Description

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

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

Note

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

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

Table SDA–4 lists the files that OpenVMS Alpha provides in SYS$LOADABLE_IMAGES that define data structure offsets.

Table SDA–9 lists the files in SYS$LOADABLE_IMAGES that define global locations within executive images.

Table SDA–9 Modules Defining Global Locations Within Executive Image

File	Contents
DDIF$RMS_EXTENSION.EXE	Support for Digital Document Interchange Format (DDIF) file operations.
ERRORLOG.STB	Error-logging routines and system services
EXCEPTION.STB	Bugcheck and exception-handling routines and those system services that declare condition and exit handlers
EXEC_INIT.STB	Initialization code
F11BXQP.STB	File system support
IMAGE_MANAGEMENT.STB	Image activator and the related system services
IO_ROUTINES.STB	$QIO system service, related system services (for example, $CANCEL and $ASSIGN), and supporting routines

(continued on next page)

Table SDA–9 (Cont.) Modules Defining Global Locations Within Executive Image

File	Contents
LOCKING.STB	Lock management routines and system services
LOGICAL_NAMES.STB	Logical name routines and system services
MESSAGE_ROUTINES.STB	System message routines and system services (including $SNDJBC and $GETTIM)
PROCESS_MANAGEMENT.STB	Scheduler, report system event, and supporting routines and system services
RECOVERY_UNIT_SERVICES.STB	Recovery unit system services
RMS.STB	Global symbols and entry points for RMS
SECURITY.STB	Security management routines and system services
SHELL*xx*K.STB	Process shell
SYS$*xx*DRIVER.EXE	Run-time device drivers
SYS$CPU_ROUTINES_*xxx*.EXE	Processor-specific data and initialization routines
SYS$NETWORK_SERVICES.EXE	DECnet support
SYS$PUBLIC_VECTORS.EXE[1]	System service vector base image
SYS$VCC.STB	Virtual I/O cache
SYS$VM.STB	System pager and swapper, along with their supporting routines, and management system services
SYSDEVICE.STB	Mailbox driver and null driver
SYSGETSYI.STB	Get System Information system service ($GETSYI)
SYSLDR_DYN.STB	Dynamic executive image loader
SYSLICENSE.STB	Licensing system service ($LICENSE)
SYSTEM_PRIMITIVES*.STB	Miscellaneous basic system routines, including those that allocate system memory, maintain system time, create fork processes, and control mutex acquisition
SYSTEM_SYNCHRONIZATION*.STB	Routines that enforce synchronization

[1]This file is located in SYS$LIBRARY.

Examples

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

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

2.
```
SDA> SHOW STACK
Process stacks (on CPU 00)
-------------------------
Current operating stack (KERNEL):

        00000000.7FF95CD0  FFFFFFFF.80430CE0  SCH$STATE_TO_COM+00040
        00000000.7FF95CD8  00000000.00000000
        00000000.7FF95CE0  FFFFFFFF.81E9CB04  LNM$SEARCH_ONE_C+000E4
        00000000.7FF95CE8  FFFFFFFF.8007A988  PROCESS_MANAGEMENT_NPRO+0E988
   SP =>00000000.7FF95CF0  00000000.00000000
        00000000.7FF95CF8  00000000.006080C1
        00000000.7FF95D00  FFFFFFFF.80501FDC
        00000000.7FF95D08  FFFFFFFF.81A5B720
   .
   .
   .
SDA>  READ/IMAGE SYS$LOADABLE_IMAGES:PROCESS_MANAGEMENT
%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]PROCESS_MANAGEMENT.STB;1
SDA>  SHOW STACK
Process stacks (on CPU 00)
-------------------------
Current operating stack (KERNEL):

        00000000.7FF95CD0  FFFFFFFF.80430CE0  SCH$FIND_NEXT_PROC
        00000000.7FF95CD8  00000000.00000000
        00000000.7FF95CE0  FFFFFFFF.81E9CB04  LNM$SEARCH_ONE_C+000E4
        00000000.7FF95CE8  FFFFFFFF.8007A988  SCH$INTERRUPT+00068
   SP =>00000000.7FF95CF0  00000000.00000000
        00000000.7FF95CF8  00000000.006080C1
        00000000.7FF95D00  FFFFFFFF.80501FDC
        00000000.7FF95D08  FFFFFFFF.81A5B720
   .
   .
   .
```

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

REPEAT

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

Format

REPEAT

Parameters

None.

Qualifiers

None.

Description

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

Example

```
SDA> SHOW CALL_FRAME
Call Frame Information
----------------------
        Stack Frame Procedure Descriptor
Flags:  Base Register = FP, Jacket, Native
        Procedure Entry: FFFFFFFF.80080CE0              MMG$RETRANGE_C+00180
        Return address on stack = FFFFFFFF.8004CF30     EXCEPTION_NPRO+00F30

Registers saved on stack
------------------------
7FF95E80 FFFFFFFF.FFFFFFFD  Saved R2
7FF95E88 FFFFFFFF.8042DBC0  Saved R3      EXCEPTION_NPRW+03DC0
7FF95E90 FFFFFFFF.80537240  Saved R4
7FF95E98 00000000.00000000  Saved R5
7FF95EA0 FFFFFFFF.80030960  Saved R6      MMG$IMGRESET_C+00200
7FF95EA8 00000000.7FF95EC0  Saved R7
7FF95EB0 FFFFFFFF.80420E68  Saved R13     MMG$ULKGBLWSL E
7FF95EB8 00000000.7FF95F70  Saved R29
   .
   .
   .
SDA> SHOW CALL_FRAME/NEXT_FP
```

```
Call Frame Information
----------------------
        Stack Frame Procedure Descriptor
Flags:  Base Register = FP, Jacket, Native
        Procedure Entry: FFFFFFFF.80F018D0              IMAGE_MANAGEMENT_PRO+078D0
        Return address on stack = FFFFFFFF.8004CF30     EXCEPTION_NPRO+00F30


Registers saved on stack
------------------------
7FF95F90 FFFFFFFF.FFFFFFFB  Saved R2
7FF95F98 FFFFFFFF.8042DBC0  Saved R3      EXCEPTION_ NPRW+03DC0
7FF95FA0 00000000.00000000  Saved R5
7FF95FA8 00000000.7FF95FC0  Saved R7
7FF95FB0 FFFFFFFF.80EF8D20  Saved R13     ERL$DEVINF O+00C20
7FF95FB8 00000000.7FFA0450  Saved R29
   .
   .
   .
SDA> REPEAT
Call Frame Information
----------------------
        Stack Frame Procedure Descriptor
Flags:  Base Register = FP, Jacket, Native
        Procedure Entry: FFFFFFFF.80F016A0              IMAGE_MANAGEMENT_PRO+076A0
        Return address on stack = 00000000.7FF2451C


Registers saved on stack
------------------------
7FFA0470 00000000.7FEEA890  Saved R13
7FFA0478 00000000.7FFA0480  Saved R29
   .
   .
   .
```

The first SHOW CALL_FRAME displays the call frame indicated by the current FP value. Because the /NEXT_FP qualifier to the instruction displays the call frame indicated by the saved FP in the current call frame, you can use the REPEAT command to repeat the SHOW CALL_FRAME/NEXT_FP command and follow a chain of call frames.

SEARCH

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

Format

SEARCH [/qualifier] range[=]expression

Parameters

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

m:n Range of locations to be searched, from *m* to *n*

m;n Range of locations to be searched, starting at *m* and continuing for *n* bytes

expression
Indication of the value for which SDA is to search. SDA evaluates the **expression** and searches the specified **range** of memory for the resulting value. For a description of SDA expressions, see Section 5.2.

Qualifiers

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

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

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

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

Description

SEARCH displays each location as each value is found. If you press Ctrl/T while using the SEARCH command, the system displays how far the search has progressed.

Examples

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

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

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

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

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

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

4.
```
SDA> SEARCH/MASK=FF000000 80000000;2000 80000000
Searching from FFFFFFFF.80000000 to FFFFFFFF.80001FFF in LONGWORD steps for 80000000...
Match at FFFFFFFF.80001000
SDA>
```

   The SEARCH command finds the value 80 in the upper byte of longword at FFFFFFFF.80001000, regardless of the contents of the lower three bytes.

SET CPU

Selects a processor to become the SDA current CPU.

Format

SET CPU cpu-id

Parameter

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

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

Qualifiers

None.

Description

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

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

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

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

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

Changing CPU context can cause an implicit change in process context under the following circumstances:

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

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

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

SET ERASE_SCREEN

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

Format

SET ERASE_SCREEN {ON|OFF}

Parameter

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

OFF
Disables the erasing of the screen.

Qualifiers

None.

Description

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

Examples

1. `SDA> SET ERASE_SCREEN ON`

   The clear screen action is now enabled.

2. `SDA>SET ERASE_SCREEN OFF`

   The clear screen action is disabled.

SET FETCH

Sets the default size of address data manipulated by the EXAMINE and EVALUATE commands.

Format

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

Parameter

QUADWORD
Sets the default size to 8 bytes.

LONGWORD
Sets the default size to 4 bytes.

WORD
Sets the default size to 2 bytes.

BYTE
Sets the default size to 1 byte.

PHYSICAL
Sets the default access method to physical addresses.

VIRTUAL
Set the default access method to virtual addresses.

Note

One and only one parameter out of each group can be specified. If both size and access method are to be changed, the two parameters should be separated by spaces and/or a comma. A comma may only be included if a parameter from both groups is specified. See examples 5 and 6.

Qualifiers

None.

Description

Sets the default size of address data manipulated by EXAMINE and EVALUATE commands. SDA uses the current default size unless it is overridden by use of the ^Q, ^L, ^W, or ^B qualifier on the @ unary operator in an expression.

This command also can set the default access method for address data manipulated by EXAMINE and EVALUATE commands. SDA uses the current default access method unless it is overridden by use of the ^P or ^V qualifier on the @ unary operator in an expression.

Examples

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

   This shows the location's contents of a 64-bit virtual address.

2.
```
SDA>SET FETCH LONG
SDA>EXAMINE @MMG$GQ_SHARED_VA_PTES
%SDA-E-NOTINPHYS, FFFFFFFF.FF7FE000 : virtual data not in physical memory
```

   This shows a failure because the SET FETCH LONG causes SDA to assume it should take the lower 32 bits of the location's contents as a longword value, sign extend them, and use that value as an address.

3.
```
SDA>EXAMINE @^QMMG$GQ_SHARED_VA_PTES
FFFFFFFD.FF7FE000:  000001D0.40001119   "...@..."
```

   This shows the correct results by overriding the SET FETCH LONG with the ^Q qualifier on the @ operator. SDA takes the full 64-bits of the location's contents and uses that value as an address.

4.
```
SDA>SET FETCH QUAD
SDA>EXAMINE @MMG$GQ_SHARED_VA_PTES
FFFFFFFD.FF7FE000:  000001D0.40001119   "...@..."
```

   This shows the correct results by changing the default fetch size to a quadword.

5.
```
SDA>SET FETCH /PHYSICAL
SDA>EXAMINE /PHYSICAL @0
```

   This command uses the contents of the physical location 0 as the physical address of the location to be examined.

6.
```
SDA>SET FETCH QUADWORD, PHYSICAL
```

   This command sets the default fetch size and default access method at the same time.

SET LOG

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

Format

SET [NO]LOG filespec

Parameter

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

Qualifiers

None.

Description

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

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

- When logging is in effect, your commands and their results are still displayed on your terminal. The SET OUTPUT command causes the displays to be redirected to the output file such that they no longer appear on the screen.
- If an SDA command requires that you press Return to produce successive screens of display, the log file produced by SET LOG will record only those screens that are actually displayed. SET OUTPUT, however, sends the entire output of all SDA commands to its listing file.
- The SET LOG command produces a log file with a default file type of .LOG; the SET OUTPUT command produces a listing file whose default file type is .LIS.
- The SET LOG command does not record output from the HELP command in its log file. The SET OUTPUT command can record HELP output in its listing file.
- The SET LOG command does not record SDA error messages in its log file. The SET OUTPUT command can record SDA error messages in its listing file.
- The SET OUTPUT command generates a table of contents, each item of which refers to a display written to its listing file. SET OUTPUT also produces running heads for each page of output. The SET LOG command does not produce these items in its log file.

Note that, if you have used the SET OUTPUT command to redirect output to a listing file, you cannot use a SET LOG command to direct the same output to a log file.

SET OUTPUT

Redirects output from SDA to the specified file or device.

Format

SET OUTPUT [/INDEX|/NOINDEX] filespec

Parameter

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

Qualifiers

/INDEX
/NOINDEX
The /INDEX qualifer causes SDA to include an index page at the beginning of the output file. This is the default. The /NOINDEX qualifier causes SDA to omit the index page from the output file.

Description

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

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

```
SDA> SET OUTPUT SYS$OUTPUT
```

If you use the SET OUTPUT command to send the SDA output to a listing file, SDA builds a table of contents that identifies the displays you selected and places the table of contents at the beginning of the output file. The SET OUTPUT command formats the output into pages and produces a running head at the top of each page.

SET PROCESS

Selects a process to become the SDA current process.

Format

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

Parameter

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

Qualifiers

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

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

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

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

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

Description

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

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

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

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

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

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

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

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

Example

```
SDA> SHOW PROCESS
Process index: 0012   Name: ERRFMT   Extended PID: 00000052
------------------------------------------------------------
Process status:  02040001   RES,PHDRES,INTER
        status2: 00000001   QUANTUM_RESCHED

PCB address              80D772C0    JIB address              80556600
PHD address              80477200    Swapfile disk address    01000F01
KTB vector address       80D775AC    HWPCB address            81260080
Callback vector address  00000000    Termination mailbox          0000
Master internal PID      00010004    Subprocess count                0
Creator extended PID     00000000    Creator internal PID     00000000
Previous CPU Id          00000000    Current CPU Id           00000000
Previous ASNSEQ  0000000000000001    Previous ASN     000000000000002E
Initial process priority        4    Delete pending count            0
# open files allowed left     100    Direct I/O count/limit        150/150
UIC               [00001,000004]    Buffered I/O count/limit      149/150
Abs time of last event   0069D34E    BUFIO byte count/limit      99424/99808
AST's remaining               247    # of threads                    1
Swapped copy of LEFC0    00000000    Timer entries allowed left     63
Swapped copy of LEFC1    00000000    Active page table count         4
Global cluster 2 pointer 00000000    Process WS page count          32
Global cluster 3 pointer 00000000    Global WS page count           31
```

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

SET RMS

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

Format

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

Parameter

option
Data structure or other information to be displayed by the SHOW PROCESS/RMS command. Table SDA–10 lists those keywords that may be used as options.

Table SDA–10 SET RMS Command Keywords for Displaying Process RMS Information

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

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

(continued on next page)

Table SDA–10 (Cont.) SET RMS Command Keywords for Displaying Process RMS Information

Keyword	Meaning
[NO]RLB	Record lock block
[NO]RU	Recovery unit structures, including the recovery unit block (RUB), recovery unit stream block (RUSB), and recovery unit file block (RUFB)
[NO]SFSB	Shared file synchronization block
[NO]WCB	Window control block
[NO]XAB	Extended attribute block
[NO]*	Current list of options displayed by the SHOW RMS command

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

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

Qualifiers

None.

Description

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

Examples

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

Display RMS structures for all IFI values.

SDA>  SET RMS=IFB
SDA>  SHOW RMS

RMS Display Options:  IFB

Display RMS structures for all IFI values.
```

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

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

RMS Display Options:  IFB,RLB,BLB,BLBSUM

Display RMS structures for all IFI values.
```

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

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

RMS Display Options:  IFB,BLB,BLBSUM
Display RMS structures only for IFI=5.
```

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

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

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

SET SIGN_EXTEND

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

Format

SET SIGN_EXTEND {ON|OFF}

Parameters

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

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

Qualifiers

None.

Description

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

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

Examples

1.
```
SDA> set sign_extend on
SDA> examine 80400000
FFFFFFFF.80400000: 23DEFF90.4A607621
```

   This shows the SET SIGN_EXTEND command as ON.

2.
```
SDA>set sign_extend off
SDA> examine 80400000
%SDA-E-NOTINPHYS, 00000000.80400000: virtual data not in physical memory
```

   This shows the SET SIGN_EXTEND command as OFF.

SHOW ADDRESS

Displays the page table related information about a memory address.

Format

SHOW ADDRESS address

Parameters

address
Displays the requested address.

Qualifier

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

Description

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

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

Examples

1. SDA> SHOW ADDRESS 80000000

```
FFFFFFFF80000000 is an S0/S1 address

Mapped by Level-3 PTE at: FFFFFFFD.FFE00000
Mapped by Level-2 PTE at: FFFFFFFD.FF7FF800
Mapped by Level-1 PTE at: FFFFFFFD.FF7FDFF0
Mapped by Selfmap PTE at: FFFFFFFD.FF7FDFF0

Also mapped in SPT window at: FFFFFFFF.FFDF0000
```

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

2. SDA> SHOW ADDRESS 0

```
00000000.00000000 is a P0 address

Mapped by Level-3 PTE at: FFFFFFFC.00000000
Mapped by Level-2 PTE at: FFFFFFFD.FF000000
Mapped by Level-1 PTE at: FFFFFFFD.FF7FC000
Mapped by Selfmap PTE at: FFFFFFFD.FF7FDFF0
```

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

3. SDA> SHOW ADDRESS FFFFFFFD.FF000000

```
FFFFFFFDFF000000 is the address of a process-private Level-2 PTE

Mapped by Level-1 PTE at: FFFFFFFD.FF7FC000
Mapped by Selfmap PTE at: FFFFFFFD.FF7FDFF0

Range mapped at level 2: FFFFFFFC.00000000 to FFFFFFFC.00001FFF (1 page)
Range mapped at level 3: 00000000.00000000 to 00000000.007FFFFF (1024 pages)
```

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

4. SDA> SHOW ADDRESS/PHYSICAL 0

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

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

5. SDA> SHOW ADDRESS/PHYSICAL 029A6000

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

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

SHOW BUGCHECK

Displays the following bugcheck codes: value, name and text.

Format

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

Parameters

name
Displays the named bugcheck code.

number
Displays the requested bugcheck code.

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

Qualifier

/ALL
Displays complete list of all the bugcheck codes and texts of number and name. It is the default.

Description

The SHOW BUGCHECK command displays the bugcheck codes that consist of value, name, and text.

Examples

1.
```
SDA> show bugcheck 100
0100  DIRENTRY       ACP failed to find same directory entry
```

   The SHOW BUGCHECK command in this example shows the requested bugcheck by number.

2.
```
SDA> show bugcheck decnet
08D0  DECNET         DECnet detected a fatal error
```

   The SHOW BUGCHECK command in this example shows the requested bugcheck by name.

3.
```
SDA> show bugcheck

BUGCHECK codes and texts
------------------------
0008  ACPMBFAIL      ACP failure to read mailbox
0010  ACPVAFAIL      ACP failure to return virtual address space
0018  ALCPHD         Allocate process header error
0020  ALCSMBCLR      ACP tried to allocate space already allocated

   .
   .
   .
```

The SHOW BUGCHECK command in this example shows the requested bugcheck by displaying all codes.

SHOW CALL_FRAME

Displays the locations and contents of the longwords representing a procedure call frame.

Format

SHOW CALL_FRAME {[starting-address]|/NEXT_FP}

Parameter

starting-address
Expression representing the starting address of the procedure call frame to be displayed. The default **starting-address** is the longword contained in the FP register of the SDA current process.

Qualifier

/NEXT_FP
Displays the procedure call frame starting at the address stored in the FP longword of the last call frame displayed by this command. You must have issued a SHOW CALL_FRAME command previously in the current SDA session in order to use the /NEXT_FP qualifier to the command.

Description

Whenever a procedure is called, information is stored on the stack of the calling routine in the form of a procedure call frame. The SHOW CALL_FRAME command displays the locations and contents of the call frame. The starting address of the call frame is determined from the specified starting address, the /NEXT_FP qualifier, or by default. The default starting address is contained in the SDA current process FP register.

When using the SHOW CALL_FRAME/NEXT_FP command to follow a chain of call frames, SDA signals the end of the chain by this message:

```
%SDA-E-NOTINPHYS, 00000000 : not in physical memory
```

This message indicates that the saved FP in the previous call frame has a zero value.

Example

```
SDA> SHOW CALL_FRAME
Call Frame Information
----------------------
        Stack Frame Procedure Descriptor
Flags:  Base Register = FP, No Jacket, Native
        Procedure Entry: FFFFFFFF.837E9F10              EXCEPTION_PRO+01F10
        Return address on stack = FFFFFFFF.837E8A1C     EXE$CONTSIGNAL_C+0019C
```

```
Registers saved on stack
------------------------
7FF95F98  FFFFFFFF.FFFFFFFB  Saved R2
7FF95FA0  FFFFFFFF.8042AEA0  Saved R3     EXCEPTION_NPRW+040A0
7FF95FA8  00000000.00000002  Saved R5
7FF95FB0  FFFFFFFF.804344A0  Saved R13    SCH$CLREF+00188
7FF95FB8  00000000.7FF9FC00  Saved R29
   .
   .
   .
SDA> SHOW CALL_FRAME/NEXT_FP
Call Frame Information
----------------------
        Stack Frame Procedure Descriptor
Flags:  Base Register = FP, No Jacket, Native
        Procedure Entry: FFFFFFFF.800FA388              RMS_NPRO+04388
        Return address on stack = FFFFFFFF.80040BFC     EXCEPTION_NPRO+00BFC

Registers saved on stack
------------------------
7FF99F60  FFFFFFFF.FFFFFFFD  Saved R2
7FF99F68  FFFFFFFF.80425BA0  Saved R3     EXCEPTION_NPRW+03DA0
7FF99F70  FFFFFFFF.80422020  Saved R4     EXCEPTION_NPRW+00220
7FF99F78  00000000.00000000  Saved R5
7FF99F80  FFFFFFFF.835C24A8  Saved R6     RMS_PRO+004A8
7FF99F88  00000000.7FF99FC0  Saved R7
7FF99F90  00000000.7FF9FDE8  Saved R8
7FF99F98  00000000.7FF9FDF0  Saved R9
7FF99FA0  00000000.7FF9FE78  Saved R10
7FF99FA8  00000000.7FF9FEBC  Saved R11
7FF99FB0  FFFFFFFF.837626E0  Saved R13    EXE$OPEN_MESSAGE+00088
7FF99FB8  00000000.7FF9FD70  Saved R29
.
    .
    .
SDA> SHOW CALL_FRAME/NEXT_FP
Call Frame Information
----------------------
        Stack Frame Procedure Descriptor
Flags:  Base Register = FP, No Jacket, Native
        Procedure Entry: FFFFFFFF.835C2438              RMS_PRO+00438
        Return address on stack = FFFFFFFF.83766020     EXE$OPEN_MESSAGE_C+00740

Registers saved on stack
------------------------
7FF9FD88  00000000.7FF9FDA4  Saved R2
7FF9FD90  00000000.7FF9FF00  Saved R3
7FF9FD98  00000000.7FFA0050  Saved R29
```

The SHOW CALL_FRAME commands in this SDA session follow a chain of call frames from that specified in the FP of the SDA current process.

SHOW CLUSTER

Displays connection manager and system communications services (SCS) information for all nodes in a cluster.

Format

SHOW CLUSTER {[{/ADDRESS=*n*|/CSID=*csid*|/NODE=*name*}]|/SCS}

Parameters

None.

Qualifiers

/ADDRESS=*n*
Displays only the OpenVMS Cluster system information for a specific OpenVMS Cluster member node, given the address of the cluster system block (CSB) for the node. This is mutually exclusive with the /CSID and /NODE qualifiers.

/CSID=*csid*
Displays only the OpenVMS Cluster system information for a specific OpenVMS Cluster member node. The value *csid* is the cluster system identification number (CSID) of the node to be displayed. You can find the CSID for a specific node in a cluster by examining the **CSB list** display of the SHOW CLUSTER command. Other SDA displays refer to a system's CSID. For instance, the SHOW LOCK command indicates where a lock is mastered or held by CSID. This is mutually exclusive with the /ADDRESS=*n* and /NODE qualifiers.

/NODE=*name*
Displays only the OpenVMS Cluster system information for a specific OpenVMS Cluster member node, given its SCS node name. This is mutually exclusive with the /ADDRESS=*n* and /CSID qualifiers.

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

Description

The SHOW CLUSTER command provides a view of the OpenVMS Cluster system from either the perspective of the connection manager (the default behavior), or from the perspective of the port driver(s) (if the /SCS qualifier is used).

OpenVMS Cluster as Seen by the Connection Manager

The SHOW CLUSTER command provides a series of displays.

The **OpenVMS Cluster summary** display supplies the following information:

- Number of votes required for a quorum
- Number of votes currently available
- Number of votes allocated to the quorum disk
- Status summary indicating whether or not a quorum is present

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

- Address of the CSB
- Name of the OpenVMS Cluster node it describes
- CSID associated with the node
- Number of votes (if any) provided by the node
- State of the CSB
- Status of the CSB

For information about the state and status of nodes, see the description of the ADD command in the *OpenVMS System Management Utilities Reference Manual*.

The **cluster block** display includes information recorded in the cluster block (CLUB), including a list of activated flags, a summary of quorum and vote information, and other data that applies to the cluster from the perspective of the node for which the SDA is being run.

The **cluster failover control block** display provides detailed information concerning the cluster failover control block (CLUFCB), and the **cluster quorum disk control block** display provides detailed information from the cluster quorum disk control block (CLUDCB).

Subsequent displays provide information for each CSB listed previously in the **CSB list** display. Each display shows the state and flags of a CSB, as well as other specific node information. (See the *OpenVMS System Management Utilities Reference Manual* for information about the flags for OpenVMS Cluster nodes.)

If any of the qualifiers /ADDRESS=*n*, /CSID=*csid*, or /NODE=name are specified, then the SHOW CLUSTER command displays only the information from the CSB of the specified node.

OpenVMS Cluster as Seen by the Port Driver

The SHOW CLUSTER/SCS command provides a series of displays.

The **SCS listening process directory** lists those processes that are listening for incoming SCS connect requests. For each of these processes, this display records the following information:

- Address of its directory entry
- Connection ID
- Name
- Explanatory information, if available

The **SCS systems summary** display provides the system block (SB) address, node name, system type, system ID, and the number of connection paths for each SCS system. An **SCS system** can be a OpenVMS Cluster member, HSC, UDA, or other such device.

Subsequent displays provide detailed information for each of the system blocks and the associated path blocks. The system block displays include the maximum message and datagram sizes, local hardware and software data, and SCS poller information. Path block displays include information that describes the connection, including remote functions and other path-related data.

Example

```
SDA> SHOW CLUSTER
OpenVMS Cluster data structures


           --- OpenVMS Cluster Summary ---

  Quorum   Votes   Quorum Disk Votes   Status Summary
  ------   -----   -----------------   --------------
     2       2            1           qf_dynvote,qf_vote,quorum

   --- CSB list ---

Address   Node    CSID      Votes   State     Status
-------   ----    ----      -----   -----     ------

805FA780  FLAM5   00010006    0     local   member,qf_same,qf_noaccess
8062C400  ROMRDR  000100ED    1     open    member,qf_same,qf_watcher,qf_active
8062C780  VANDQ1  000100EF    0     open    member,qf_same,qf_noaccess

   --- Cluster Block (CLUB) 805FA380 ---

Flags: 16080005 cluster,qf_dynvote,init,qf_vote,qf_newvote,quorum

Quorum/Votes                  2/2    Last transaction code          02
Quorum Disk Votes               1    Last trans. number            596
Nodes                           3    Last coordinator CSID    000100EF
Quorum Disk               $1$DIA0    Last time stamp       31-DEC-1992
Found Node SYSID     00000000FC03                              17:26:35
Founding Time          3-JAN-1993    Largest trans. id        00000254
                         21:04:21    Resource Alloc. retry           0
Index of next CSID           0007    Figure of Merit          00000000
Quorum Disk Cntrl Block  805FADC0    Member State Seq. Num        0203
Timer Entry Address      00000000    Foreign Cluster          00000000
CSP Queue                   empty
  --- Cluster Failover Control Block (CLUFCB) 805FA4C0 ---

Flags: 00000000

Failover Step Index      00000037    CSB of Synchr. System    8062C780
Failover Instance ID     00000254


  --- Cluster Quorum Disk Control Block (CLUDCB) 805FADC0 ---

State     : 0002 qs_rem_act
Flags     : 0100 qf_noaccess
CSP Flags : 0000

Iteration Counter               0            UCB address     00000000
Activity Counter                0            TQE address     805FAE00
Quorum file LBN          00000000            IRP address     00000000
                                             Watcher CSID    000100ED


  --- FLAM5 Cluster System Block (CSB) 805FA780 ---

State:  0B local
Flags:  070260AA member,qf_same,qf_noaccess,selected,local,status_rcvd,send_status
Cpblty: 00000000
SWVers: 7.0
HWName: DEC 3000 Model 400
```

```
Quorum/Votes       1/0    Next seq. number    0000    Send queue      00000000
Quor. Disk Vote      1    Last seq num rcvd   0000    Resend queue    00000000
CSID          00010006    Last ack. seq num   0000    Block xfer Q.   805FA7D8
Eco/Version       0/23    Unacked messages       0    CDT address     00000000
Reconn. time  00000000    Ack limit              0    PDT address     00000000
Ref. count           2    Incarnation   1-JAN-1993    TQE address     00000000
Ref. time  31-AUG-1992                    00:00:00    SB address      80421580
              17:26:35    Lock mgr dir wgt       0    Current CDRP    00000001

  --- ROMRDR Cluster System Block (CSB) 8062C400 ---

State:  01 open
Flags:  0202039A member,qf_same,cluster,qf_active,selected,status_rcvd
Cpblty: 00000000
SWVers: 7.0
HWName: DEC 3000 Model 400

Quorum/Votes       2/1    Next seq. number    B350    Send queue      00000000
Quor. Disk Vote      1    Last seq num rcvd   E786    Resend queue    00000000
CSID          000100ED    Last ack. seq num   B350    Block xfer Q.   8062C458
Eco/Version       0/22    Unacked messages       1    CDT address     805E8870
Reconn. time  00000000    Ack limit              3    PDT address     80618400
Ref. count           2    Incarnation  19-AUG-1992    TQE address     00000000
Ref. time  19-AUG-1992                    16:15:00    SB address      8062C140
              16:17:08    Lock mgr dir wgt       0    Current CDRP    00000000

  --- VANDQ1 Cluster System Block (CSB) 8062C780 ---

State:  01 open
Flags:  020261AA member,qf_same,qf_noaccess,cluster,selected,status_rcvd
Cpblty: 00000000
SWVers: 7.0
HWName: DEC 3000 Model 400

Quorum/Votes       1/0    Next seq. number    32B6    Send queue      00000000
Quor. Disk Vote      1    Last seq num rcvd   A908    Resend queue    00000000
CSID          000100EF    Last ack. seq num   32B6    Block xfer Q.   8062C7D8
Eco/Version       0/23    Unacked messages       1    CDT address     805E8710
Reconn. time  00000000    Ack limit              3    PDT address     80618400
Ref. count           2    Incarnation  17-AUG-1992    TQE address     00000000
Ref. time  19-AUG-1992                    15:37:06    SB address      8062BCC0
              16:21:22    Lock mgr dir wgt       0    Current CDRP    00000000

         --- SWPCTX Cluster System Block (CSB) 80D3B1C0 ---

State:  0B local
Flags:  030A60AA member,qf_same,qf_noaccess,selected,send_ext_status,local,status_rcvd
Cpblty: 00000037 rm8sec,vcc,dts,cwcreprc,threads
SWVers: V7.0
HWName: DEC 3000 Model 400

Quorum/Votes       1/1    Next seq. number    0000    Send queue      00000000
Quor. Disk Vote      1    Last seq num rcvd   0000    Resend queue    00000000
CSID          00010001    Last ack. seq num   0000    Block xfer Q.   80D3B218
Eco/Version       0/26    Unacked messages       0    CDT address     00000000
Reconn. time  00000000    Ack limit              0    PDT address     00000000
Ref. count           2    Incarnation  12-JUL-1996    TQE address     00000000
Ref. time  16-JUL-1996                    15:36:17    SB address      80C50800
              16:15:48    Lock mgr dir wgt       0    Current CDRP    00000001
```

This example illustrates the default output of the SHOW CLUSTER command.

SHOW CONNECTIONS

Displays information about all active connections between System Communications Services (SCS) processes or a single connection.

Format

SHOW CONNECTIONS [{/ADDRESS=*cdt-address*|/NODE=*name*|/SYSAP=*name*}]

Parameters

None.

Qualifiers

/ADDRESS=*cdt-address*
Displays information contained in the connection descriptor table (CDT) for a specific connection. You can find the **cdt-address** for any active connection on the system in the *CDT summary page* display of the SHOW CONNECTIONS command. In addition, CDT addresses are stored in many individual data structures related to SCS connections. These data structures include class driver request packets (CDRPs) and unit control blocks (UCBs) for class drivers that use SCS, and cluster system blocks (CSBs) for the connection manager.

/NODE=*name*
Displays all CDTs associated with the specified remote SCS node name.

/SYSAP=*name*
Displays all CDTs associated with the specified local SYSAP.

Description

The SHOW CONNECTIONS command provides a series of displays.

The **CDT summary page** lists information regarding each connection on the local system, including the following:

- CDT address
- Name of the local process with which the CDT is associated
- Connection ID
- Current state
- Name of the remote node (if any) to which it is currently connected

The **CDT summary page** concludes with a count of CDTs that are free and available to the system.

SHOW CONNECTIONS next displays a page of detailed information for each active CDT listed previously.

Example

```
SDA> SHOW CONNECTIONS

    --- CDT Summary Page ---

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

 805E7ED0     SCS$DIRECTORY       FF120000        listen
 805E8030     MSCP$TAPE           FF120001        listen
 805E8190     VMS$VMScluster      FF120002        listen
 805E82F0     MSCP$DISK           FF120003        listen
 805E8450     SCA$TRANSPORT       FF120004        listen
 805E85B0     MSCP$DISK           FF150005        open         VANDQ1
 805E8710     VMS$VMScluster      FF120006        open         VANDQ1
 805E8870     VMS$VMScluster      FF120007        open         ROMRDR
 805E89D0     MSCP$DISK           FF120008        open         ROMRDR
 805E8C90     VMS$DISK_CL_DRVR    FF12000A        open         ROMRDR
 805E8DF0     VMS$DISK_CL_DRVR    FF12000B        open         VANDQ1
 805E8F50     VMS$TAPE_CL_DRVR    FF12000C        open         VANDQ1

Number of free CDT's:  188

          --- Connection Descriptor Table (CDT) 80C44850 ---

State:  0001 listen              Local Process:        MSCP$TAPE
Blocked State:  0000

Local Con. ID   899F0003    Datagrams sent         0    Message queue   80C4488C
Remote Con. ID  00000000    Datagrams rcvd         0    Send Credit Q.  80C44894
Receive Credit         0    Datagram discard       0    PB address      00000000
Send Credit            0    Message Sends          0    PDT address     00000000
Min. Rec. Credit       0    Message Recvs          0    Error Notify    822FFCC0
Pend Rec. Credit       0    Mess Sends NoFP        0    Receive Buffer  00000000
Initial Rec. Credit    0    Mess Recvs NoFP        0    Connect Data    00000000
Rem. Sta.   000000000000    Send Data Init.        0    Aux. Structure  00000000
Rej/Disconn Reason     0    Req Data Init.         0    Fast Recvmsg Rq 00000000
Queued for BDLT        0    Bytes Sent             0    Fast Recvmsg PM 00000000
Queued Send Credit     0    Bytes rcvd             0    Change Affinity 00000000
                            Total bytes map        0

   --- Connection Descriptor Table (CDT) 805E8030 ---

State:  0001 listen    Local Process:        MSCP$TAPE
Blocked State:  0000

Local Con. ID   FF120001    Datagrams sent         0    Message queue   805E8060
Remote Con. ID  00000000    Datagrams rcvd         0    Send Credit Q.  805E8068
Receive Credit         0    Datagram discard       0    PB address      00000000
Send Credit            0    Messages Sent          0    PDT address     00000000
Min. Rec. Credit       0    Messages Rcvd.         0    Error Notify    804540D0
Pend Rec. Credit       0    Send Data Init.        0    Receive Buffer  00000000
Initial Rec. Credit    0    Req Data Init.         0    Connect Data    00000000
Rem. Sta.   000000000000    Bytes Sent             0    Aux. Structure  00000000
Rej/Disconn Reason     0    Bytes rcvd             0
Queued for BDLT        0    Total bytes map        0
Queued Send Credit     0
   .
   .
   .
```

This example shows the default output of the SHOW CONNECTIONS command.

SHOW CPU

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

Format

SHOW CPU [cpu-id]

Parameter

cpu-id
Numeric value from 00 to $1F_{16}$ indicating the identity of the processor for which context information is to be displayed. If you specify a value outside this range, or you specify the **cpu-id** of a processor that was not active at the time of the system failure, SDA displays the following message:

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

If you use the **cpu-id** parameter, the SHOW CPU command performs an implicit SET CPU command, making the processor indicated by **cpu-id** the current CPU for subsequent SDA commands. (See the description of the SET CPU command and Section 4 for information on how this can affect the CPU context—and process context—in which SDA commands execute.)

Qualifiers

None.

Description

The SHOW CPU command displays system failure information about the processor specified by **cpu-id** or, by default, the SDA current CPU, as defined in Section 4. You cannot use the SHOW CPU command when examining the running system with SDA.

The SHOW CPU command produces several displays. First, there is a brief description of the system failure and its environment that includes the following:

- Reason for the bugcheck.
- Name of the currently executing process. If no process has been scheduled on this processor, SDA displays the following message:

  ```
  Process currently executing: no processes currently scheduled on the processor
  ```

- File specification of the image executing within the current process (if there is a current process).
- Interrupt priority level (IPL) of the processor at the time of the system failure.

Next, the **general registers** display shows the contents of the processor's integer registers (R0 to R30), and the AI, RA, PV, FP, PC, and PS at the time of the system failure.

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

- Common processor registers
- Processor-specific registers
- Stack pointers

The first part of the processor registers display includes registers common to all Alpha processors, which are used by the operating system to maintain the current process virtual address space, system space, or other system functions. This part of the display includes the following registers:

- Hardware privileged context block base register (PCBB)
- System control block base register (SCBB)
- Software interrupt summary register (SISR)
- Address space number register (ASN)
- AST summary register (ASTSR)
- AST enable register (ASTEN)
- Interrupt priority level register (IPL)
- Processor priority level register (PRBR)
- Page table base register (PTBR)
- Virtual page table base register (VPTB)
- Floating point control register (FPCR)
- Machine check error summary register (MCES)

The last part of the display includes the four stack pointers: the pointers of the kernel, executive, supervisor, and user stacks (KSP, ESP, SSP, and USP, respectively).

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

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

Example

```
SDA> SHOW CPU
CPU 00 Processor crash information



CPU 00 reason for Bugcheck: UNXINTEXC, Unexpected interrupt or exception

Process currently executing on this CPU: UETCLIG00master

Current image file: $1$DKB400:[SYS64.SYSCOMMON.][SYSTEST]UETCLIG00.EXE;1

Current IPL: 13  (decimal)

CPU database address:  805AE000

General registers:

R0  = 00000000.00000001 R1  = 00000000.0000003B R2  = FFFFFFFF.8004FF88
R3  = FFFFFFFF.80428070 R4  = 00000000.00000001 R5  = 00000000.00000D04
R6  = 00000000.7FF78BE6 R7  = 00000000.00000064 R8  = FFFFFFFF.806CEA96
R9  = 00000000.00000030 R10 = 00000000.00002270 R11 = 00000000.0C040087
R12 = 00000000.00000001 R13 = FFFFFFFF.80435270 R14 = FFFFFFFF.80434AE0
R15 = FFFFFFFF.80403200 R16 = 00000000.00000410 R17 = 00000000.00000001
R18 = 00000000.000005D0 R19 = 00000000.000000EA R20 = FFFFFFFF.80403200
R21 = FFFFFFFF.8040C810 R22 = 00000000.000000FA R23 = FFFFFFFF.8040C7F0
R24 = FFFFFFFF.8040C7E0 AI  = 00000000.00000000 RA  = 00000000.00000014
PV  = 00000000.0000003B R28 = 00000000.0000003B FP  = 00000000.7FF95D00
PC  = FFFFFFFF.80050020 PS  = 00000000.00000D04


Processor Internal Registers:

ASN  = 00000000.00000000                     ASTSR/ASTEN =          00000000
IPL  =          00000008  PCBB = 00000000.0140C080  PRBR = FFFFFFFF.80C0C000
PTBR = 00000000.000000B8  SCBB = 00000000.00000250  SISR = 00000000.00000000
VPTB = FFFFFFFC.00000000  FPCR = 00000000.00000000  MCES = 00000000.00000000

        KSP     = 00000000.7FF95A00
        ESP     = 00000000.7FF9A000
        SSP     = 00000000.7FFA04C0
        USP     = 00000000.7EE719F0

                 Spinlocks currently owned by CPU 00

SCHED                              ADDRESS   80427880
Ownership Depth    00000001        Rank      00000012
CPUs Waiting       00000000        Index     00000032
```

This example shows the default output of the SHOW CPU command.

SHOW CRASH

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

Format

SHOW CRASH [/CPU=*n*]

Parameters

None.

Qualifier

/CPU=n
Allows exception data to be displayed from CPUs other than the one considered as the crash CPU when more than one CPU crashes simultaneously.

Description

The SHOW CRASH command has two different manifestations, depending on whether it is issued in the analysis of a running system or a system failure.

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

When used during the analysis of a running system, the SHOW CRASH command produces a display that describes the system and the version of OpenVMS Alpha that it is running. The **system crash information** display contains the following information:

- Date and time that the ANALYZE/SYSTEM command was issued (titled "Time of system crash" in the display)
- Name and version number of the operating system
- Major and minor IDs of the operating system
- Identity of the Alpha system, including an indication of its cluster membership
- CPU ID of the primary CPU
- Exception display for fatal system bugchecks or PGFIPLHI bugchecks

When used during the analysis of a system failure, the SHOW CRASH command produces several displays that identify the system and describe its state at the time of the failure.

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

- Date and time of the system failure.
- Name and version number of the operating system.

- Major and minor IDs of the operating system.
- Identity of the system.
- CPU IDs of both the primary CPU and the CPU that initiated the bugcheck. In an Alpha uniprocessor system, these IDs are identical.
- For each active processor in the system, the name of the bugcheck that caused the system failure. Generally, there will be only one significant bugcheck in the system. All other processors typically display the following as their reason for taking a bugcheck:

  ```
  CPUEXIT, Shutdown requested by another CPU
  ```

Subsequent screens of the SHOW CRASH command display information about the state of each active processor on the system at the time of the system failure. The information in these screens is identical to that produced by the SHOW CPU command, including the general-purpose registers, processor-specific registers, stack pointers, and records of spinlock ownership. The first such screen presents information about the processor that caused the failure; others follow according to the numeric order of their CPU IDs.

Examples

1.
```
SDA> SHOW CRASH
System crash information


Time of system crash: 24-JAN-1995 10:16:12.71

Version of system: OpenVMS Alpha VERSION 7.0

System Version Major ID/Minor ID: 1/0

System type: Flamingo/EV4

Crash CPU ID/Primary CPU ID:  00/00

Bitmask of CPUs active/available:  00000001/00000001

CPU bugcheck codes:
        CPU 00 -- SSRVEXCEPT, Unexpected system service exception

System State at Time of Exception
---------------------------------
Exception Frame:
----------------
 R2  = 00000000.00001200
 R3  = FFFFFFFF.80425BA0
 R4  = FFFFFFFF.80422020
 R5  = FFFFFFFF.80444C88
 R6  = 00000000.7FFD0080
 R7  = 00000000.00000000
 PC  = FFFFFFFF.8010D480
 PS  = 30000000.0000000A

%SYSTEM-F-ACCVIO, access violation, reason mask=00, virtual address=0000000000000008,
    PC=FFFFFFFF8010D480, PS=0000000A
```

```
Saved Registers in Mechanism Array
----------------------------------
R0   = 00000000.7FFD01E8  R1   = 00000000.00000000  R16  = 00000000.7FFD008C
R17  = 00000000.00000001  R18  = 00000000.00000000  R19  = 00000000.00000000
R20  = 00000000.00000001  R21  = 00000000.7FFF0140  R22  = 00000000.00000002
R23  = 00000000.00000008  R24  = 00000000.00000000  R25  = 00000000.00000003
R26  = FFFFFFFF.8010974C  R27  = 00000000.000001FF  R28  = 00000000.000001FF

CPU 00 reason for Bugcheck: SSRVEXCEPT, Unexpected system service exception

Process currently executing on this CPU: SERVER_001C

Current IPL: 0  (decimal)

CPU database address:  805AE000

General registers:

R0   = 00000000.00000004  R1   = FFFFFFFF.80405C30  R2   = 00000000.00001200
R3   = FFFFFFFF.80425BA0  R4   = FFFFFFFF.80422020  R5   = FFFFFFFF.80444C88
R6   = 00000000.7FFD0080  R7   = 00000000.00000000  R8   = 00000000.7FF9FDF0
R9   = 00000000.00000000  R10  = 00000000.00000002  R11  = 00000000.7FFD0080
R12  = 00000000.00000008  R13  = FFFFFFFF.8044DB78  R14  = 00000000.7FFD0080
R15  = 00000000.7FEE1C20  R16  = 00000000.000003C0  R17  = 00000000.7FF99C80
R18  = 00000000.7FF99E40  R19  = FFFFFFFF.80425F28  R20  = 00000000.00000001
R21  = 00000000.7FFF0140  R22  = FFFFFFFF.8335C000  R23  = 00000000.7FF9A000
R24  = 00000000.7FFF0028  AI   = 00000000.00000002  RA   = FFFFFFFF.837E9F3C
PV   = FFFFFFFF.80405C30  R28  = FFFFFFFF.837E8810  FP   = 00000000.7FF99C10
PC   = FFFFFFFF.80002010  PS   = 00000000.00000009


Processor Internal Registers:

ASN  = 00000000.00000000                    ASTEN/ASTSR =          0000000F
IPL  =          00000000  PCBB = 00000000.02F28080  PRBR = FFFFFFFF.805AE000
PTBR = 00000000.000012DA  SCBB = 00000000.00000500  SISR = 00000000.00000000
VPTB = FFFFFFFC.00000000  FPCR = 00000000.00000000  MCEX = 00000000.00000000

        KSP    = 00000000.7FF96000
        ESP    = 00000000.7FF99BF8
        SSP    = 00000000.7FF9FD70
        USP    = 00000000.7FE6B780

                No spinlocks currently owned by CPU 00
```

This long display reflects the output of the SHOW CRASH command within the analysis of a system failure.

2.
```
SDA> SHOW CRASH
System crash information


Time of system crash: 19-JAN-1995 10:16:12.71

Version of system: OpenVMS Alpha VERSION 7.0

System Version Major ID/Minor ID: 1/0

System type: Flamingo/EV4

Crash CPU ID/Primary CPU ID:  00/00

Bitmask of CPUs active/available:  00000001/00000001
```

```
CPU bugcheck codes:
        CPU 00 -- PGFIPLHI, Page fault with IPL too high

System State at Time of Page Fault:
---------------------------------

Page fault for address 00000000 7FFAB000 occured at IPL: 18
Memory management flags: 80000000 00000000 (data write)

Exception Frame:
----------------
        R2  = 00000000.7FFF0200
        R3  = 00000000.00000000
        R4  = FFFFFFFF.805DC700
        R5  = 00000000.7FF8C000
        R6  = FFFFFFFF.808C4F40
        R7  = 00000000.00000000
        PC  = FFFFFFFF.80BB4A2C  EXE$PRCDELMSG_C+005FC
        PS  = 30000000.00000200

        FFFFFFFF.80BB4A1C:      BLE             R0,#X000009
        FFFFFFFF.80BB4A20:      BIS             R31,R1,R17
        FFFFFFFF.80BB4A24:      ADDQ            R2,#X04,R16
        FFFFFFFF.80BB4A28:      BIS             R31,R0,R25
  PC => FFFFFFFF.80BB4A2C:      INSQUEL/D
        FFFFFFFF.80BB4A30:      LDQ             R24,#X0078(R13)
        FFFFFFFF.80BB4A34:      BIS             R31,R25,R0
        FFFFFFFF.80BB4A38:      SUBL            R0,#X01,R0
        FFFFFFFF.80BB4A3C:      ADDL            R1,R24,R1

   PS =>
         MBZ SPAL      MBZ    IPL VMM MBZ CURMOD INT PRVMOD
         0   30   00000000000 02  0   0   KERN   0   KERN
```

This display reflects the output of a SHOW CRASH command within the analysis of a PGFIPLHI bugcheck.

SHOW DEVICE

Displays a list of all devices in the system and their associated data structures, or displays the data structures associated with a given device or devices.

Format

SHOW DEVICE {[device-name] | /ADDRESS=ucb-address}

Parameter

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

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

Qualifier

/ADDRESS=ucb-address
Indicates the device for which data structure information is to be displayed by the address of its unit control block (UCB). The /ADDRESS qualifier is an alternate method of supplying a device name to the SHOW DEVICE command. If both the **device-name** parameter and the /ADDRESS qualifier appear in a single SHOW DEVICE command, SDA responds only to the parameter or qualifier that appears first.

Description

The SHOW DEVICE command produces several displays taken from system data structures that describe the devices in the system configuration.

If you use the SHOW DEVICE command to display information for more than one device or one or more controllers, it initially produces the **DDB (device data block) list** display to provide a brief summary of the devices for which it renders information in subsequent screens.

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

- Address of the device data block (DDB)
- Controller name
- Name of the ancillary control process (ACP) associated with the device
- Name of the device driver
- Address of the driver prologue table (DPT)

The SHOW DEVICE command then produces a display of information pertinent to the device controller. This display includes information gathered from the following structures:

- Device data block (DDB)
- Primary channel request block (CRB)
- Interrupt dispatch block (IDB)
- Driver dispatch table (DDT)

If the controller is an HSC controller, SHOW DEVICE also displays information from its system block (SB) and each path block (PB).

Many of these structures contain pointers to other structures and driver routines. Most notably, the DDT display points to various routines located within driver code, such as the start I/O routine, unit initialization routine, and cancel I/O routine.

For each device unit subject to the SHOW DEVICE command, SDA displays information taken from its unit control block, including a list of all I/O request packets (IRPs) in its I/O request queue. For certain mass storage devices, SHOW DEVICE also displays information from the primary class driver data block (CDDB), the volume control block (VCB), and the ACP queue block (AQB). For units that are part of a shadow set, SDA displays a summary of shadow set membership.

As it displays information for a given device unit, SHOW DEVICE defines the following symbols as appropriate:

Symbol	Meaning
UCB	Address of unit control block
SB	Address of system block
ORB	Address of object rights block
DDB	Address of device data block
DDT	Address of driver dispatch table
CRB	Address of channel request block
AMB	Associated mailbox UCB pointer
IRP	Address of I/O request packet
2P_UCB	Address of alternate UCB for dual-pathed device
LNM	Address of logical name block for mailbox
PDT	Address of port descriptor table

Symbol	Meaning
CDDB	Address of class driver descriptor block for MSCP served device
2P_CDDB	Address of alternate CDDB for MSCP served device
RWAITCNT	Resource wait count for MSCP served device
VCB	Address of volume control block for mounted device

If you are examining a driver-related system failure, you may find it helpful to issue a SHOW STACK command after the appropriate SHOW DEVICE command, examining the stack for any of these symbols. Note, however, that although the SHOW DEVICE command defines those symbols relevant to the last device unit it has displayed, and redefines symbols relevant to any subsequently displayed device unit, it does not undefine symbols. (For instance, SHOW DEVICE DUA0 defines the symbol PDT, but SHOW DEVICE MBA0 does not undefine it, even though the PDT structure is not associated with a mailbox device.) In order to maintain the accuracy of such symbols that appear in the stack listing, use the DEFINE command to modify the symbol name. For example:

```
SDA> DEFINE DUA0_PDT PDT
SDA> DEFINE MBA0_UCB UCB
```

See the descriptions of the READ and FORMAT commands for additional information on defining and examining the contents of device data structures.

Examples

1.
```
SDA>SHOW DEVICE/ADDRESS=8041E540
OPA0                            VT300_Series       UCB address    8041E540

Device status:   00000010 online
Characteristics: 0C040007 rec,ccl,trm,avl,idv,odv
                 00000200 nnm

Owner UIC [000001 ,000004]  Operation count        160    ORB address    8041E4E8
      PID         00010008  Error count              0    DDB address    8041E3F8
Class/Type           42/70  Reference count          2    DDT address    8041E438
Def. buf. size          80  BOFF              00000001    CRB address    8041E740
DEVDEPEND         180093A0  Byte count        0000012C    I/O wait queue 8041E5AC
DEVDEPND2         FB101000  SVAPTE            80537B80
DEVDEPND3         00000000  DEVSTS            00000001
FLCK index              3A
DLCK address      8041E880

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

This example reproduces the SHOW DEVICE display for a single device unit, OPA0. Whereas this display lists information from the UCB for OPA0, including some addresses of key data structures and a list of pending I/O requests for the unit, it does not display information about the controller or its device driver. To display the latter information, specify the **device-name** as OPA (for example, SHOW DEVICE OPA).

2. SDA> SHOW DEVICE DU
I/O data structures

```
------------------

                           DDB list
                           --------

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

      80D0B3C0    BLUES$DUA     F11XQP    SYS$DKDRIVER          807735B0
      8000B2B8    RED$DUA       F11XQP    SYS$DKDRIVER          807735B0
      80D08BA0    BIGTOP$DUA    F11XQP    SYS$DKDRIVER          807735B0
      80D08AE0    TIMEIN$DUA    F11XQP    SYS$DKDRIVER          807735B0
   .
   .
   .
Press RETURN for more.
   .
   .
   .
```

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

SHOW DUMP

Displays formatted information of the header, error log buffers, logical memory blocks (LMBs), compression data, and dump summary. It can also be used to display hexadecimal information of individual blocks.

Format

SHOW DUMP {/ALL | /BLOCK[=*m*[{: | ;}*n*]] | [/COMPRESSION_MAP[=*m*[:*n*]]
| /ERROR_LOGS | /HEADER | /LMB[={*ALL* | *n*}] | /SUMMARY]}

Parameter

None

Qualifiers

/ALL
Displays the equivalent to specifying all the /SUMMARY, /HEADER, /ERROR_LOGS, /COMPRESSION_MAP, and /LMB=*ALL* qualifiers.

/BLOCK[=*m*[{: | ;}*n*]]
Displays a hexadecimal dump of one or more blocks. Ranges can be expressed by using the following syntax:

no value	Displays next block
m	Displays single block
m:n	Displays a range of blocks from *m* to *n*, inclusive
m;n	Displays a range of blocks starting at *m* and continuing for *n* blocks

/COMPRESSION_MAP[=*m*[:*n*]]
Displays details of the compression data. Levels of detail can be expressed by using the following syntax:

no value	Displays a summary of all compression map blocks
m	Displays contents of a single compression map block
m:n	Displays details of single compression map entry

/ERROR_LOGS
Displays a summary of the error log buffers.

/HEADER
Displays the formatted contents of the dump header.

/LMB[={*ALL* | *n*}]
Displays the formatted contents of logical memory block (LMB) headers and the virtual address (VA) ranges within the LMB. LMBs to be displayed can be expressed by using the following syntax:

no value	Displays next LMB
n	Displays LMB at block *n* of the dump
ALL	Displays all LMBs

/SUMMARY
Displays a summary of the dump. This is the default.

Description

The SHOW DUMP command displays information about the structure of the dump file. It displays the header, the error log buffers, the compression map, and in the case of a selective dump, the logical memory block (LMB) headers. This command is provided for use when troubleshooting dump analysis problems.

Example

```
SDA >SHOW DUMP/SUMMARY

Summary of dump file DKA300:[SYS0.SYSEXE]SYSDUMP.DMP;8
------------------------------------------------------
Dump type:                  Compressed selective
Size of dump file:          000203A0/000203A0 (132000./132000.)
Highest VBN written:        0000D407          (54279.)
Uncompressed equivalent:    0001AF1C          (110364.)
Compression ratio:          2.03:1            (49.2%)

                                                                 Uncomp    Uncomp
                     Dump file section             VBN     Blocks    VBN     blocks
-----------------------------------------------  ---------- --------  -------  --------
Dump header                                      00000001 00000002
Error log buffers                                00000003 00000020
Compression map                                  00000023 00000010
LMB 0000 (PT space)                              00000033 00000038  00000033 000000D2
LMB 0001 (S0/S1 space)                           0000006B 0000621B  00000105 000095A5
LMB 0002 (S2 space)                              00006286 000001A3  000096AA 00000352
LMB 0003 (Page tables of key process "SYSTEM")   00006429 00000005  000099FC 00000062
LMB 0004 (Memory of key process "SYSTEM")        0000642E 00000071  00009A5E 00000342
   .
   .
   .
LMB 0003 (Page tables of key process "NETACP")   0000697B 00000009  0000AE14 00000052
LMB 0004 (Memory of key process "NETACP")        00006984 000013F7  0000AE66 00001F42
LMB 0005 (Key global pages)                      00007D7B 000002BA  0000CDA8 00000312
LMB 0006 (Page tables of process "DTWM")         00008035 00000013  0000D0BA 00000082
LMB 0007 (Memory of process "DTWM")              00008048 000013A3  0000D13C 000022E4
   .
   .
   .
LMB 0006 (Page tables of process "Milord_FTA1:") 0000C5E3 00000005  00019A44 00000062
LMB 0007 (Memory of process "Milord_FTA1:")      0000C5E8 00000074  00019AA6 00000222
LMB 0008 (Remaining global pages)                0000C65C 00000DAC  00019CC8 00001255
```

This example of the SHOW DUMP/SUMMARY command gives a summary of the dump.

```
SDA> SHOW DUMP/HEADER

Dump header
-----------

    Header field                    Meaning                       Value

-------------------  -------------------------------------  -----------------
```

```
DMP$W_FLAGS           Flags                                                0FC1
                       DMP$V_OLDDUMP:    Dump has been analyzed
                       DMP$V_WRITECOMP:  Dump write was completed
                       DMP$V_ERRLOGCOMP: Error log buffers written
                       DMP$V_DUMP_STYLE: Selective dump
                                         Verbose messages
                                         Dump off system disk
                                         Compressed
DMP$B_FLAGS2          Additional flags                                       09
                       DMP$V_COMPRESSED: Dump is compressed
                       DMP$V_ALPHADUMP:  This is an OpenVMS Alpha dump

DMP$Q_SYSIDENT        System version                                  "X69G-FT1"
DMP$Q_LINKTIME        Base image link date/time       " 8-JUN-1996 02:07:27.31"
DMP$L_SYSVER          Base image version                               03000000
DMP$W_DUMPVER         Dump version                                        0704

DMP$L_DUMPBLOCKCNT    Count of blocks dumped for memory                0000D3D5
DMP$L_NOCOMPBLOCKCNT  Uncompressed blocks dumped for memory            0001AEEA
DMP$L_SAVEPRCCNT      Number of processes saved                        00000014

   .
   .
   .

EMB$Q_CR_TIME         Crash date/time                 " 3-JUL-1996 09:30:13.36"
EMB$L_CR_CODE         Bugcheck code                                 "SSRVEXCEPT"
EMB$B_CR_SCS_NAME     Node name                                       "SWPCTX  "
EMB$T_CR_HW_NAME      Model name                          "DEC 3000 Model 400"
EMB$T_CR_LNAME        Process name                                    "SYSTEM"

DMP$L_CHECKSUM        Dump header checksum                             439E5E91
```

This example of the SHOW DUM/HEADER command shows the information in the header.

SHOW EXECUTIVE

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

Format

SHOW EXECUTIVE

Parameters

None.

Qualifiers

None.

Description

The executive consists of two base images and a number of other executive images.

The base image called SYS$BASE_IMAGE.EXE contains:

- Symbol vectors for universal executive routines and data cells
- Procedure descriptors for universal executive routines
- Globally referenced data cells

The base image called SYS$PUBLIC_VECTORS.EXE contains:

- Symbol vectors for system service procedures
- Procedure descriptors for system services
- Transfer routines for system services

The base images are the pathways to routines and system service procedures in the other executive images.

The SHOW EXECUTIVE command lists the location and size of each executive image. It can enable you to determine whether a given memory address falls within the range occupied by a particular image. (Table SDA–9 describes the contents of each executive image.)

SHOW EXECUTIVE also displays the nonzero length image section base address and length. The base address and length are blank for sliced loadable executive images.

By default, SDA displays each location within an executive image as an offset from the beginning of one of the loadable images; for instance, EXCEPTION+00282. Similarly, those symbols that represent system services point to the transfer routine in SYS$PUBLIC_VECTORS.EXE and not to the actual system service procedure. When tracing the course of a system failure through the listings of modules contained within a given executive image, you may find it useful to load into the SDA symbol table all global symbols and global entry points defined within one or all executive images. See the description of the READ command for additional information.

The SHOW EXECUTIVE command usually shows all components of the executive, as illustrated in the following example. In rare circumstances, you may obtain a partial listing. For instance, once it has loaded the EXCEPTION module (in the INIT phase of system initialization), the system can successfully post a bugcheck exception and save a crash dump before loading all the executive images normally loaded.

Example

```
SDA> SHOW EXECUTIVE
OpenVMS Alpha Executive Layout
------------------------------
Image                              Base            End             Length       SymVec

SYSWSDRIVER
      Nonpaged read only          802DE000        802DF400        00001400
      Nonpaged read/write         80CB2600        80CB2E00        00000800
      Linked 1-OCT-1995 13:07       LDRIMG        80DEEA00
SYS$IKDRIVER
      Nonpaged read only          802D2000        802DC800        0000A800
      Nonpaged read/write         80CB1000        80CB2600        00001600
      Linked 1-OCT-1995 13:56       LDRIMG        80DE9840
SYS$IMDRIVER
      Nonpaged read only          802CC000        802D0A00        00004A00
      Nonpaged read/write         80CB0400        80CB1000        00000C00
      Linked  1-OCT-1995 13:56      LDRIMG        80DE9580
SYS$INDRIVER
      Nonpaged read only          802BC000        802CAA00        0000EA00
      Nonpaged read/write         80CAF400        80CB0400        00001000
      Linked  1-OCT-1995 13:57      LDRIMG        80DE9100
SYS$RTTDRIVER
      Nonpaged read only          802B8000        802BB600        00003600
      Nonpaged read/write         80CAEA00        80CAF400        00000A00
      Linked 30-SEP-1995 22:17      LDRIMG        80DE4A00
SYS$CTDRIVER
      Nonpaged read only          802AC000        802B6C00        0000AC00
      Nonpaged read/write         80CACE00        80CAEA00        00001C00
      Linked 30-SEP-1995 22:10      LDRIMG        80DE4440
NDDRIVER
      Nonpaged read only          802A8000        802AB600        00003600
      Nonpaged read/write         80CAC400        80CAC300        00000A00
      Linked 30-SEP-1995 22:14      LDRIMG        80D143CO
NETDRIVER
      Nonpaged read only          80290000        802A7800        00017800
      nonpaged read/write         80CA9A00        80CAC400        00002A00
      Paged read only             8028E000        8028E200        00000200
      Linked 30-SEP-1995 22:12      LDRIMG        80D13E80
SYS$SODRIVER
      Nonpaged read only          8028A000        8028DC00        00003C00
      Nonpaged read/write         80CA8800        80CA9A00        00001200
      Linked 30-SEP-1995 22:14      LDRIMG        80DBEAC0
SYS$YRDRIVER
      Nonpaged read only          80282000        80288200        00006200
```

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

SHOW GLOBAL_SECTION_TABLE

Displays information contained in the global section table.

Format

SHOW GLOBAL_SECTION_TABLE or SHOW GST [/QUALIFIER]

Parameter

None

Qualifiers

/SECTION_INDEX=n
Displays only the global section table entry for the specified section.

Description

Displays the entire contents of the global section table, unless the qualifier /SECTION_INDEX is specified. This command is equivalent to SHOW PROCESS /PROCESS_SECTION_TABLE/SYSTEM. See the SHOW PROCESS command and Table SDA–26 for more information.

```
SDA> SHOW GST

Global Section Table
--------------------

Global section table information
--------------------------------

        Last entry allocated     0187
        First free entry         0000

Global section table
--------------------
INDEX  ADDRESS    SECT/GPTE ADDR     PAGELETS   WINDOW     VBN      CCB/GSD    REFCNT   FLINK BLINK   FLAGS
0001   80D09FD8  FFFFFFFF.82A24000   00000069  80D202C0  00000003  00000000  00000007  0000  0000
0002   80D09FB0  FFFFFFFF.82FE0000   00000160  80D73B80  00000428  00000000  00000016  0000  0000
0003   80D09F88  FFFFFFFF.82A5A000   0000005F  80D206C0  0000014F  00000000  00000006  0000  0000
0004   80D09F60  FFFFFFFF.829A8000   00000001  80D73B80  0000058B  00000000  00000001  0000  0000   WRT CRF
0005   80D09F38  FFFFFFFF.82A6E000   00000009  80D21080  00000027  00000000  00000001  0000  0000
0006   80D09F10  FFFFFFFF.82998000   00000008  80D73D00  00000005  00000000  00000001  0000  0000
0007   80D09EE8  FFFFFFFF.82A76000   0000009B  80D21240  0000015A  00000000  0000000A  0000  0000
0008   80D09EC0  FFFFFFFF.829B0000   00000013  80D73EC0  00000003  00000000  00000002  0000  0000
000A   80D09E70  FFFFFFFF.8300C000   00000228  80D74080  00000002  00000000  00000015  0000  0000   WRT CRF
000B   80D09E48  FFFFFFFF.82AB0000   00000012  80D25280  000000A0  00000000  00000002  0000  0000
000C   80D09E20  FFFFFFFE.00052010   000001C2  80D88900  0000006F  81782030  00000000  000C  000C   GBL
                                     NAME = INS$81781FC0_003
000D   80D09DF8  FFFFFFFF.82ABA000   00000059  80D26880  00000043  00000000  00000006  0000  0000
000E   80D09DD0  FFFFFFFE.00052108   00000021  80D90E40  0000000E  81782EB0  00000000  000E  000E   GBL
                                     NAME = INS$81782E30_003
000F   80D09DA8  FFFFFFFF.82ACE000   00000025  80D27E40  00000022  00000000  00000003  0000  0000
0010   80D09D80  FFFFFFFE.00052130   00000058  80D90F80  0000001B  81783280  00000000  0010  0010   GBL
                                     NAME = INS$81783210_003
0011   80D09D58  FFFFFFFF.82ADA000   000001C7  80D2B100  00000046  00000000  0000001D  0000  0000
0012   80D09D30  FFFFFFFE.00052170   000000AE  80D91BC0  00000038  81783690  00000000  0012  0012   GBL
                                     NAME = INS$81783620_003
0013   80D09D08  FFFFFFFF.82B22000   00000029  80D2C6C0  00000007  00000000  00000003  0000  0000
0014   80D09CE0  FFFFFFFE.000521D8   0000002F  80D92000  0000000E  81783A80  00000000  0014  0014   GBL
                                     NAME = INS$81783A10_003
0015   80D09CB8  FFFFFFFE.00052200   00000161  80D92300  000000B4  81783EA0  00000000  0015  0015   GBL
                                     NAME = INS$81783E20_003
0016   80D09C90  FFFFFFFF.82B36000   0000005C  80D2E440  00000024  00000000  00000006  0000  0000
0017   80D09C68  FFFFFFFE.000522C8   00000170  80D92300  00000267  81783EF0  00000000  0017  0017   GBL
                                     NAME = INS$81783E20_008
0018   80D09C40  FFFFFFFF.82B46000   000000AB  80D2FA00  0000006B  00000000  0000000B  0000  0000
   .
   .
   .
```

ZK-8829A-GE

SHOW GSD

Displays information contained in the global section descriptors.

Format

SHOW GSD [/QUALIFIERS]

Parameter

None

Qualifiers

/ADDRESS=*n*
Displays a specific global section descriptor entry, given its address.

/ALL
Displays information in all the global section descriptors; that is, the system, group, and deleted global section descriptors. This qualifier is the default.

/SYSTEM
Displays information in the system global section descriptors.

/GROUP
Displays information in the group global section descriptors.

/DELETED
Displays information in the deleted (that is, delete pending) global section descriptors.

Description

The SHOW GSD displays information that resides in the global section descriptors. Table SDA–11 shows the fields and their meaning.

Table SDA–11 GSD Fields

Field	Meaning
ADDRESS	Gives the address of the global section descriptor.
NAME	Gives the name of the global section.
GSTX	Gives the global section table index.
FLAGS	Gives the settings of flags for specified global section, as a hexadecimal number, then key flag bits are also displayed by name.
BASEPFN†	Gives physical page frame number at which the section starts.
PAGES†	Gives number of pages (not pagelets) in section.
REFCNT†	Gives number of times this global section is mapped.

†This field only applies to PFN mapped global sections.

```
SDA > SHOW GSD

System Global Section Descriptor List
-------------------------------------

                                                                                  ----------PFNMAP----------
--
ADDRESS     NAME          GSTX  FLAGS     WRT AMOD=USER PERM                      BASEPFN   PAGES     REFCNT

817DAF30    SECIDX_422    02DD  0082C3C9  DZRO WRT AMOD=USER PAGFIL
817DAE60    SECIDX_421    02DC  008A83CD  DZRO WRT AMOD=USER PERM PAGFIL
817DAD90    SECDIX_420    02DB  0088C3CD  DZRO WRT AMOD=USER PAGFIL
817DACC0    SECDIX_419    02DA  008883DC  AMOD=USER PERM                          00000B0B  00000002  00000000
817DABE0    SECIDX_418    0000  0001C3C1  AMOD=USER PERM                          00000B0B  00000002  00000000
817DAB00    SECIDX_417    0000  0001C3C1  DZRO WRT AMOD=USER PERM
817DA890    SECIDX_412    02D6  0080C3CD  DZRO WRT AMOD=USER
817DA850    SECIDX_411    02D5  008083CD
    .
    .
    .
```

ZK-8830A-GE

SHOW HEADER

Displays the header of the dump file.

Format

SHOW HEADER

Parameters

None.

Qualifiers

None.

Description

The SHOW HEADER command produces a 10-column display, each line of which displays both the hexadecimal and ASCII representation of the contents of the dump file header in 32-byte intervals. Thus, the first eight columns, when read right to left, represent the hexadecimal contents of 32 bytes of the header; the ninth column, when read left to right, records the ASCII equivalent of the contents. (Note that the period [.] in this column indicates an ASCII character that cannot be displayed.)

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

See the *OpenVMS AXP Internals and Data Structures* manual for a discussion of the information contained in the dump file header.

```
SDA> SHOW HEADER

Dump file header
----------------
00000000 7FFA6000 00000000 7FFA1C98 00000000 0000187C 08090FC1 00000004  ....Á...|.........ú......‘ú.....     00000000
00001FFF 0000000D 00002000 80D0A000 00000000 7AFFBAD0 00000000 7FFAC100  .Áú.....Đº.z......Đ.............     00000020
0000B162 00000000 00000001 00000000 00040704 FCFFFFFF 03000000 80C13670  p6Á.......ü.............b±..     00000040
00000000 00000400 00000008 00000000 3154462D 31393658 00000011 00000000  ........X691-FT1................     00000060
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................     00000080
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ................................     000000A0
FF7FC000 FFFFFFFD FF000000 80C220F0 00000000 00000000 00000000 00000000  ............ð Â.....ý....À......     000000C0
.
.
.


Saved error log messages
------------------------
0004FFF9 0000040B 00000001 00000000 00000070 80D0B000 80D0A00C 00000000  ......Đ..°Đ.p.......ù...........     80D0A000
B4510020 60030000 00000000 00000020 20585443 50575308 00000000 00020000  ......... SWPCTX  ......‘ .Q´....     80D0A020
30303320 43454412 00000002 00000000 3154462D 31393658 0000009A 2C31075A  Z.1,....X691-FT1.........DEC 300     80D0A040
000000AA 59EC7C0A 00000000 00000000 00000000 00303034 206C6564 6F4D2030  0 Model 400.............|ìYª...     80D0A060
20585443 50575308 00000000 00020000 0004FFF9 0000040B 00000001 00000000  ............ù...........SWPCTX     80D0A080
3154462D 31393658 0001009A 2C3107FD 1DDB0040 60030000 00000000 00000020   .........‘@.Û.ý.1,....X691-FT1     80D0A0A0
00000000 00303034 206C6564 6F4D2030 30303320 43454412 00000003 00000000  .........DEC 3000 Model 400.....     80D0A0C0
4B442458 54435057 530A0064 000001AB 00000000 00010001 00000000 00000000  ................«...d..SWPCTX$DK     80D0A0E0
.
.
.                                                                                                      ZK-8861A-GE
```

The SHOW HEADER command displays the contents of the dump file's header. Ellipses indicate hexadecimal information omitted from the display.

SHOW LAN

Displays information contained in various local area network (LAN) data structures.

Format

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

Parameters

None.

Qualifiers

/CLIENT=*name*
Specifies that information be displayed for the specified client. Valid client designators are SCA, DECNET, LAT, MOPRC, TCPIP, DIAG, ELN, BIOS, LAST, USER, ARP, MOPDL, LOOP, BRIDGE, DNAME, ENCRY, DTIME, and LTM. The /CLIENT, /DEVICE, and /UNIT qualifiers are synonymous and mutually exclusive.

/CLUEXIT
Specifies that cluster protocol information be displayed.

/COUNTERS
Specifies that the LAN station block (LSB) and unit control block (UCB) counters be displayed.

/CSMACD
Specifies that Carrier Sense Multiple Access with Collision Detect (CSMA/CD) information for the LAN be displayed. By default, both CSMA/CD and Fiber Distributed Data Interface (FDDI) information is displayed.

/DEVICE=*name*
Specifies that information be displayed for the specified device, unit, or client. For each LAN adapter on the system there is one **device** and multiple users of that device called **units** or **clients**. Device designators are specified in the format **XXdn**, where **XX** is the type of device, **d** is the device letter, and **n** is the unit number. The device letter and unit number are optional. The first unit, which is always present, is the template unit. These are specified as indicated in this example, for a DEMNA which is called EX:

/DEVICE=EX—display all EX devices on the system
/DEVICE=EXA—display the first EX device only
/DEVICE=EXA0—display the first EXA unit
/DEVICE=SCA—display SCA unit
/DEVICE=LAT—display LAT units

Valid client names are listed in the /CLIENT=name qualifier. The /CLIENT, /DEVICE, and /UNIT qualifiers are synonymous and mutually exclusive.

/ELAN
Specifies information from an Emulated LAN (ELAN) that runs over an asynchronous transfer mode (ATM) network. The /ELAN qualifier displays the LAN station Block (LSB) address, device state, and the LSB fields pertinent

to an ELAN for both the parent ATM device and the ELAN psuedo-device drivers. It also specifies the name, description, parent device, state, and LAN emulation client (LEC) attributes of the ELAN.

The qualifier /ELAN used with the device qualifier (/LAN/device=ELA) will only display information for the specified device or psuedo-device.

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

/FDDI
Specifies that Fiber Distributed Data Interface (FDDI) information for the LAN be displayed. By default, both CSMA/CD and FDDI information is displayed.

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

/ICOUNTERS
Specifies internal counters of the drivers by displaying the internal counters. If the /ICOUNTERS qualifier is used with the /DEVICE qualifier, the /ICOUNTERS specifies the internal counters of a specific driver.

/QUEUE
Specifies a listing of all queues, whether their status is valid or invalid, and all elements of the queues. If the /QUEUE qualifier is used with the /DEVICE qualifier, the /QUEUE specifies a specific queue.

/SUMMARY
Specifies that only a summary of LAN information (a list of flags, LSBs, UCBs, and base addresses) be printed. This is the default.

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

/UNIT=*name*
Specifies that information be displayed for the specified unit. See the descriptions for /CLIENT=name and /DEVICE=name qualifiers.

/VCI
Specifies the VMS Communication Interface Block (VCIB) for each LAN device with an active VCI user. If the /VCI qualifier is used with the /DEVICE qualifier, the VCIB is only displayed for the specified device.

Description

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

Examples

1. SDA> SHOW LAN/FULL

```
LAN Data Structures
-------------------
               -- LAN Information Summary 23-MAY-1996 13:07:52 --

LAN flags:  00000004 LAN_INIT

LAN block address           80DB7140    Timer DELTA time            10000000
Number of stations                 2    DAT sequence number                1
LAN module version                 1    First SVAPTE                FFDF60F0
LANIDEF version                   51    Number of PTEs                     3
LANUDEF version                   26    SVA of first page           8183C000
First LSB address           80DCA980

            -- LAN CSMACD Network Management 23-MAY-1996 13:07:52 --

Creation time                    None    Times created                      0
Deletion time                    None    Times deleted                      0
Module EAB                  00000000    Latest EIB                  00000000
Port EAB                    00000000
Station EAB                 00000000
NM flags:   00000000

             -- LAN FDDI Network Management 23-MAY-1996 13:07:52 --

Creation time                    None    Times created                      0
Deletion time                    None    Times deleted                      0
Module EAB                  00000000    Link EAB                    00000000
Port EAB                    00000000    PHY port EAB                00000000
Station EAB                 00000000    Module EIB                  00000000
NM flags:   00000000

LAN Data Structures
-------------------
               -- ESA Device Information 23-MAY-1996 13:07:52 --

LSB address                 80DCA980    Driver code address         80CAE838
Driver version     00000001.07010037    Device1 code address        00000000
Device1 version    00000000.00000000    Device2 code address        00000000
Device2 version    00000000.00000000    LAN code address            80CAFA00
LAN version        00000001.07010112    DLL type                      CSMACD
Device name                EY_NITC2    MOP name                         MXE
MOP ID                           94    HW serial               Not supplied
HW version                 00000000    Promiscuous mode                OFF
Controller mode              NORMAL    Promiscuous UCB             00000000
Internal loopback               OFF    All multicast state             OFF
Hardware address   08-00-03-DE-00-12    CRC generation mode              ON
Physical address   AA-00-04-00-88-FE    Full Duplex Enable              OFF
Active unit count                  1    Full Duplex State               OFF
Line speed                        10

Flags:  00000000
Char:   00000000
Status: 00000003 RUN,INITED

LAN Data Structures
-------------------
            -- ESA Device Information (cont) 23-MAY-1996 13:07:52 --
```

```
Put rcv ptr/index          00000000    Get rcv ptr/index          00000015
Put xmt ptr/index          80DCB620    Get xmt ptr/index          80DCB620
Put cmd ptr/index          00000000    Get cmd ptr/index          00000000
Put uns ptr/index          00000000    Get uns ptr/index          00000000
Put smt ptr/index          00000000    Get smt ptr/index          00000000
RBufs owned by dev                0    Rcv packet limit                 32
XEnts owned by dev                0    XEnts owned by host               4
CEnts owned by dev                0    Transmit timer                   0
UEnts owned by dev                0    Control timer                    0
SEnts owned by dev                0    Periodic SYSID timer           599
Current rcv buffers              17    Ring unavail timer               0
Rqst MAX rcv buffers             32    USB timer                       26
Rqst MIN rcv buffers             16    Receive alignment                0
Curr MAX rcv buffers             32    Receive buffer size           1518
Curr MIN rcv buffers             16    Min 1st chain segment            0
FILL rcv buffers                 16    Min transmit length              0
ADD  rcv buffers                 32    Dev xmt header size              0

LAN Data Structures
-------------------

            -- ESA Device Information (cont) 23-MAY-1996 13:07:52 --

Last receive         23-MAY 13:07:51    Last transmit        23-MAY 13:07:50
ADP address                80D4B280    IDB address                80DCA880
DAT stage                  00000000    DAT xmt status     0000003C.003C0001
DAT number started                1    DAT xmt complete     23-MAY 13:07:19
DAT number failed                 0    DAT rcv found                  None
DAT VCRP                   80DCBB80    DAT UCB                    00000000
Mailbox enable flag               0    CRAM read comman           00000000
CSR base phys addr 00000000.00000000    CRAM write comma           00000000
Mailboxes in use                  0    Media                          UNDF
2nd LW status flags        00000000

LAN Data Structures
-------------------
         -- ESA Network Management Information 23-MAY-1996 13:07:52 --

Creation time                 None    Create count                     0
Deletion time                 None    Enable count                     0
Enabled time                  None    Number of ports                  0
Disabled time                 None    Events logged                    0
EIB address                00000000    NMgmt assigned addr         None
LLB address                00000000    Station name itmlst     00000000
LHB address                00000000    Station itmlst len               0
First LPB address          00000000

LAN Data Structures
-------------------
                -- ESA Fork Information 23-MAY-1996 13:07:52 --

ISR   FKB sched      23-MAY 13:07:51    ISR   FKB in use flag        FREE
ISR   FKB time       23-MAY 13:07:51    ISR   FKB count               200
IPL8  FKB sched      23-MAY 13:07:20    IPL8  FKB in use flag        FREE
IPL8  FKB time       23-MAY 13:07:20    IPL8  FKB count                 1
RESET FKB sched             None    RESET FKB in use flag        FREE
RESET FKB time              None    RESET FKB count                  0
NM    FKB sched             None    NM    FKB in use flag        FREE
NM    FKB time              None    NM    FKB count                  0
Fork status code               0
```

```
LAN Data Structures
-------------------
                -- ESA Queue Information 23-MAY-1996 13:07:52 --
Control hold queue      80DCACF0  Status:  Valid, empty
Control request queue   80DCACF8  Status:  Valid, empty
Control pending queue   80DCAD00  Status:  Valid, empty
Transmit request queue  80DCACE8  Status:  Valid, empty
Transmit pending queue  80DCAD18  Status:  Valid, empty
Receive buffer list     80DCAD38  Status:  Valid, 17 elements
Receive pending queue   80DCAD20  Status:  Valid, empty
Post process queue      80DCAD08  Status:  Valid, empty
Delay queue             80DCAD10  Status:  Valid, empty
Auto restart queue      80DCAD28  Status:  Valid, empty
Netwrk mgmt hold queue  80DCAD30  Status:  Valid, empty

          -- ESA Multicast Address Information 23-MAY-1996 13:07:52 --

AB-00-00-04-00-00

                  -- ESA Unit Summary 23-MAY-1996 13:07:52 --

UCB      UCB Addr  Fmt   Value           Client     State
---      --------  ---   -----           ------  -----------
ESA0     80D4F6C0
ESA1     80E35400  Eth   60-03           DECNET  0017 STRTN,LEN,UNIQ,STRTD

LAN Data Structures
-------------------
              -- ESA Counters Information 23-MAY-1996 13:07:52 --

Octets received                 596    Octets sent                     230
PDUs received                     8    PDUs sent                         5
Mcast octets received           596    Mcast octets sent               138
Mcast PDUs received               8    Mcast PDUs sent                   3
Unrec indiv dest PDUs             0    PDUs sent, deferred               0
Unrec mcast dest PDUs             1    PDUs sent, one coll               0
Data overruns                     0    PDUs sent, mul coll               0
Unavail station buffs             0    Excessive collisions              0
Unavail user buffers              0    Late collisions                   0
CRC errors                        0    Carrier check failure             0
Alignment errors                  0    Last carrier failure           None
Rcv data length err               0    Coll detect chk fail              5
Frame size errors                 0    Short circuit failure             0
Frames too long                   0    Open circuit failure              0
Seconds since zeroed             34    Transmits too long                0
Station failures                  0    Send data length err              0

LAN Data Structures
-------------------
           -- ESA Counters Information (cont) 23-MAY-1996 13:07:52 --
```

```
No work transmits                 0    Ring avail transitions            0
Buffer_Addr transmits             0    Ring unavail transitions          0
SVAPTE/BOFF transmits             0    Loopback sent                     0
Global page transmits             0    System ID sent                    0
Bad PTE transmits                 0    ReqCounters sent                  0
Restart pending counter           0    Internal counters size           40
+00 MCA not enabled             187    +2C Generic (or unused)     00000000
+04 Xmt underflows                0    +30 Generic (or unused)     00000000
+08 Rcv overflows                 0    +34 Generic (or unused)     00000000
+0C Memory errors                 0    +38 Generic (or unused)     80DCAD18
+10 Babbling errors               0    +3C Generic (or unused)     80DCAD18
+14 Local buffer errors           0    +40 Generic (or unused)     004E0840
+18 LANCE interrupts            202    +44 Generic (or unused)     61616161
+1C Xmt ring <31:0>         00000000    +48 Generic (or unused)     61616161
+20 Xmt ring <63:32>        00000000    +4C Generic (or unused)     61616161
+24 Soft errors handled           0    +50 Generic (or unused)     61616161
+28 Generic (or unused)     00000000    +54 Generic (or unused)     61616161

LAN Data Structures
-------------------
               -- ESA Error Information 23-MAY-1996 13:07:52 --

Fatal error count                 0    Last error CSR              00000000
Fatal error code               None    Last fatal error                None
Prev  error code               None    Prev fatal error                None
Transmit timeouts                 0    Last USB time                   None
Control timeouts                  0    Last UUB time                   None
Restart failures                  0    Last CRC time                   None
Power failures                    0    Last CRC srcadr                 None
Bad PTE transmits                 0    Last length erro                None
Loopback failures                 0    Last exc collisi                None
System ID failures                0    Last carrier fai                None
ReqCounters failures              0    Last late collis                None

LAN Data Structures
-------------------
             -- ESA0 Template Unit Information 23-MAY-1996 13:07:52 --

LSB address                80DCA980    Error count                       0
VCIB address               00000000    Parameter mask             00000000
Stop IRP address           00000000    Promiscuous mode                OFF
Restart IRP address        00000000    All multicast mode              OFF
LAN medium                   CSMACD    Source Routing mode      TRANSPARENT
Packet format              Ethernet    Access mode                EXCLUSIVE
Eth protocol type             00-00    Shared user DES                 None
802E protocol ID      00-00-00-00-00    Padding mode                     ON
802.2 SAP                        00    Automatic restart           DISABLED
802.2 Group SAPs         00,00,00,00    Allow prom client                ON
Controller mode               NORMAL    Can change address              OFF
Internal loopback                OFF    802.2 service                  User
CRC generation mode               ON    Rcv buffers to save               1
Functional Addr mod               ON    Minimum rcv buffers               4
Hardware address   08-00-03-DE-00-12    User transmit FC/AC              ON
Physical address   FF-FF-FF-FF-FF-FF    User receive FC/AC              OFF

LAN Data Structures
-------------------
          -- ESA1 60-03 (DECNET) Unit Information 23-MAY-1996 13:07:52 --
```

```
LSB address                 80DCA980    Error count                        0
VCIB address                00000000    Parameter mask              00DA8695
Stop IRP address            80E047C0    Promiscuous mode                 OFF
Restart IRP address         00000000    All multicast mode               OFF
LAN medium                    CSMACD    Source Routing mode       TRANSPARENT
Packet format               Ethernet    Access mode                EXCLUSIVE
Eth protocol type              60-03    Shared user DES                 None
802E protocol ID      00-00-00-00-00    Padding mode                     ON
802.2 SAP                       00      Automatic restart            DISABLED
802.2 Group SAPs         00,00,00,00    Allow prom client                ON
Controller mode               NORMAL    Can change address               OFF
Internal loopback                OFF    802.2 service                   User
CRC generation mode               ON    Rcv buffers to save               10
Functional Addr mod               ON    Minimum rcv buffers                4
Hardware address   08-00-03-DE-00-12    User transmit FC/AC               ON
Physical address   AA-00-04-00-88-FE    User receive FC/AC                OFF

LAN Data Structures
-------------------
      -- ESA1 60-03 (DECNET) Unit Information (cont) 23-MAY-1996 13:07:52 --

Last receive         23-MAY 13:07:47    Starter's PID               0001000F
Last transmit        23-MAY 13:07:50    Maximum header size               16
Last start attempt   23-MAY 13:07:20    Maximum buffer size             1498
Last start done      23-MAY 13:07:20    Rcv quota charged              15040
Last start failed               None    Default FC value                  00
MCA match enabled                 01    Default AC value                  00
Last MCA filtered  AB-00-00-04-00-00    Maintenance state                 ON

UCB status:  00000017 STRTN,LEN,UNIQ,STRTD

Receive IRP queue       80E356E8  Status:  Valid, 1 element
Receive pending queue   80E356E0  Status:  Valid, empty

Multicast address table, embedded:
  AB-00-00-04-00-00

LAN Data Structures
-------------------
        -- ESA1 60-03 (DECNET) Counters Information 23-MAY-1996 13:07:52 --

Octets received                 483    Octets sent                      180
PDUs received                     7    PDUs sent                          3
Mcast octets received           483    Mcast octets sent                180
Mcast PDUs received               7    Mcast PDUs sent                    3
Unavail user buffer               0    Multicast not enabled              0
Last UUB time                  None    User buffer too small              0
```

The SHOW LAN/FULL command displays information for all LAN, LSB, and UCB data structures.

2. SDA> SHOW LAN/TIME

```
                -- LAN History Information 12-FEB-1995 11:08:48 --
```

```
12-FEB 11:08:47.92  ESA              Last receive
12-FEB 11:08:47.92  ESA              Last fork scheduled
12-FEB 11:08:47.92  ESA              Last fork time
12-FEB 11:08:47.77  ESA5     LAST     Last receive
12-FEB 11:08:47.72  ESA3     LAT      Last receive
12-FEB 11:08:41.25  ESA              Last transmit
12-FEB 11:08:41.25  ESA5     LAST     Last transmit
12-FEB 11:08:40.02  ESA2     DECnet   Last receive
12-FEB 11:08:39.14  ESA2     DECnet   Last transmit
12-FEB 11:08:37.39  ESA3     LAT      Last transmit
12-FEB 10:19:25.31  ESA              Last unavail user buffer
12-FEB 10:19:25.31  ESA2     DECnet   Last unavail user buffer
11-FEB 14:10:20.09  ESA5     LAST     Last start completed
11-FEB 14:10:02.16  ESA3     LAT      Last start completed
11-FEB 14:09:58.44  ESA2     DECnet   Last start completed
11-FEB 14:09:57.44  ESA              Last DAT transmit
```

The SHOW LAN/TIME command displays print time information from device and unit data structures.

3. SDA>SHOW LAN/VCI/DEVICE=ICB

```
               -- ICB VCI Information 17-APR-1996 14:22:07 --

LSB address  = 80A1D580
Device state = 00000003 RUN,INITED

         -- ICB2 80-41 (LAST) VCI Information 17-APR-1996 14:22:07 --

VCIB address =    8096F238
CLIENT flags:     00000001 RCV_DCB
LAN flags:        00000004 LAN_INIT
DLL flags:        00000005 XMT_CHAIN,PORT_STATUS
UCB status:       00000015 STRTN,UNIQ,STRTD

VCI ID                        LAST    VCI version              00010001
UCB address                80A4C5C0    DP VCRP address          00000000
Hardware address  00-00-93-08-52-CF    LDC address              80A1D720
Physical address  00-00-93-08-52-CF    LAN medium                     TR
Transmit available         80A1D670    Outstanding operations          0
Maximum receives                 0    Outstanding receives            0
Max xmt size                  4444    Header size                    52
Build header rtn           808BF230    Report event rtn         86327130
XMT initiate rtn           808BF200    Transmit complete rtn    86326D80
XMT frame rtn              808BF210    Receive complete rtn     86326A80

      -- ICB2 80-41 (LAST) VCI Information (cont) 17-APR-1996 14:22:07 --

Portmgmt initiate rtn      808BF0C0    Portmgmt complete rtn    86327100
Monitor request rtn        00000000    Monitor transmit rtn     00000000
Monitor flags              00000000    Monitor receive rtn      00000000
Port usable                00000000    Port unusable            00000000
```

The SHOW LAN/VCI/DEVICE=ICB command displays the VCIB for a Token Ring device (ICB) which has an active VCI user (LAST).

4. SDA>SHOW LAN/ELAN

```
        -- HCA Emulated LAN LSB Information 17-APR-1996 14:08:02 --

LSB address = 8098D200
Device state = 00000101 RUN,RING_AVAIL
```

```
Driver CM VC setup adr   808986A0    Driver CM VC teardown adr    80898668
NIPG CM handle adr       8096C30C    NIPG CM SVC handle           00000000
NIPG CM agent handle adr 809B364C    NIPG CM mgr lineup handle    809B394C
NIPG CM ILMI IO handle   809B378C    MIB II handle adr            809B94CC
MIB handle adr           809B3ACC    Queue header for EL LSBs     00000000
DEC MIB handle adr       809BBD8C    NIPG current TQEs used       00000000
Count of allocated TQEs  0000000D    NIPG current pool used       0000D2C0
NIPG pool allocations    00075730

        -- ELA Emulated LAN LSB Information 17-APR-1996 14:08:02 --

LSB address = 80AB08C0
Device state = 00000001 RUN

ELAN name = ELAN 1
ELAN description = ATM ELAN
ELAN parent = HCA0
ELAN state = 00000001 ACTIVE

MAX transmit size     MTU_1516          ELAN media type      LAN_802_3
LEC attr buff adr     80AB1FC0          LEC attr buff size    00000328
Event mask            00000000          PVC identifer         00000000
Extended sense        00000000

         -- ELA Emulated LAN LEC Attributes 17-APR-1996 14:08:02 --

LAN type          00000000          LAN MTU              00000001
Proxy flag        00000000          Control timeout      0000000A
Max UF count      00000001          Max UF time          00000001
VCC timeout       000004B0          Max retry count      00000002
LEC id            00000002          Forw delay time      0000000F
Flush timeout     00000004          Path switch delay    00000006
SM state          00000070          Illegal CTRL frames  00000000
CTRL xmt failures 00000000          CTRL frames sent     0000000C
CTRL frames_rcvd  00000012          LEARPs sent          00000000
LEARPS rcvd       00000000          UCASTs sent direct   00000000
UCASTs flooded    00000006          UCASTs discarded     00000001
NUCASTs sent      00000000
Local ESI         00000000.00000000
BUS ATM addr      39999900000000008002BA57E80.AA000302FF12.00
LES ATM addr      39999900000000008002BA57E80.AA000302FF14.00
My ATM addr       39999900000000008002BA57E80.08002B2240A0.00
```

The SHOW LAN/ELAN command displays information for the parent ATM device (HCA) driver and the ELAN psuedo-device (ELA) driver.

5.
```
SDA>SHOW LAN/ELAN/DEV=ELA

        -- ELA Emulated LAN LSB Information 17-APR-1996 14:08:22 --

LSB address = 80AB08C0
Device state = 00000001 RUN

ELAN name = ELAN 1
ELAN description = ATM ELAN
ELAN parent = HCA0
ELAN state = 00000001 ACTIVE

MAX transmit size     MTU_1516          ELAN media type      LAN_802_3
LEC attr buff adr     80AB1FC0          LEC attr buff size    00000328
Event mask            00000000          PVC identifer         00000000
Extended sense        00000000

         -- ELA Emulated LAN LEC Attributes 17-APR-1996 14:08:22 --
```

```
LAN type             00000000          LAN MTU              00000001
Proxy flag           00000000          Control timeout      0000000A
Max UF count         00000001          Max UF time          00000001
VCC timeout          000004B0          Max retry count      00000002
LEC id               00000002          Forw delay time      0000000F
Flush timeout        00000004          Path switch delay    00000006
SM state             00000070          Illegal CTRL frames  00000000
CTRL xmt failures    00000000          CTRL frames sent     0000000C
CTRL frames_rcvd     00000012          LEARPs sent          00000000
LEARPS rcvd          00000000          UCASTs sent direct   00000000
UCASTs flooded       00000006          UCASTs discarded     00000001
NUCASTs sent         00000000
Local ESI            00000000.00000000
BUS ATM addr         39999900000000008002BA57E80.AA000302FF12.00
LES ATM addr         39999900000000008002BA57E80.AA000302FF14.00
My ATM addr          39999900000000008002BA57E80.08002B2240A0.00
```

The SHOW LAN/ELAN/DEVICE=ELA command displays information for the ELAN psuedo-device (ELA) driver only.

6.
```
SDA> SHOW LAN/ELAN/DEVICE=HCA

       -- HCA Emulated LAN LSB Information 17-APR-1996 14:08:25 --

LSB address = 8098D200
Device state = 00000101 RUN,RING_AVAIL

Driver CM VC setup adr   808986A0    Driver CM VC teardown adr    80898668
NIPG CM handle adr       8096C30C    NIPG CM SVC handle           00000000
NIPG CM agent handle adr 809B364C    NIPG CM mgr lineup handle    809B394C
NIPG CM ILMI IO handle   809B378C    MIB II handle adr            809B94CC
MIB handle adr           809B3ACC    Queue header for EL LSBs     00000000
DEC MIB handle adr       809BBD8C    NIPG current TQEs used       00000000
Count of allocated TQEs  0000000D    NIPG current pool used       0000D2C0
NIPG pool allocations    000757B2
```

The SHOW LAN/ELAN/DEVICE=HCA command displays information for the ATM device (HCA) driver only.

SHOW LOCK

Displays information about all lock management locks in the system, or about a specified lock.

Format

SHOW LOCK {lock-id|/ADDRESS=*n*|/ALL (d)|/CACHED |/NAME=*name*}

Parameter

lock-id
Name of a specific lock.

Qualifiers

/ADDRESS=*n*
Displays a specific lock, given the address of the lock block.

/ALL
Lists all locks that exist in the system. This is the default behavior of the SHOW LOCK command.

/CACHED
Displays locks that are no longer valid. The memory for these locks is kept around so that later requests for locks can use them. Cached locks are not displayed in the other SHOW LOCK commands.

/NAME=*name*
Displays a specified lock with the given name.

Description

The SHOW LOCK command displays the information described in Table SDA–12 for each lock management lock in the system, or for the lock indicated by **lock-id**. (Use the SHOW SPINLOCKS command to display information about spin locks.) You can obtain a similar display for the locks owned by a specific process by issuing the appropriate SHOW PROCESS/LOCKS command. See the *OpenVMS Programming Concepts Manual* for additional information.

You can display information about the resource to which a lock is queued by issuing the SHOW RESOURCE command specifying the resource's **lock-id**.

Table SDA–12 Contents of the SHOW LOCK and SHOW PROCESS/LOCKS Displays

Display Element	Description
Process Index[1]	Index in the PCB array to a pointer to the process control block (PCB) of the process that owns the lock.
Name[1]	Name of the process that owns the lock.

[1]This display element is produced only by the SHOW PROCESS/LOCKS command.

(continued on next page)

Table SDA–12 (Cont.) Contents of the SHOW LOCK and SHOW PROCESS /LOCKS Displays

Display Element	Description
Extended PID[1]	Clusterwide identification of the process that owns the lock.
Lock ID	Identification of the lock.
PID	Systemwide identification of the lock.
Flags	Information specified in the request for the lock.
Par. ID	Identification of the lock's parent lock.
Granted at	Lock mode at which the lock was granted.
Sublocks	Identification numbers of the locks that the lock owns.
LKB	Address of the lock block (LKB). If a blocking AST has been enabled for this lock, the notation "BLKAST" appears next to the LKB address.
Resource	Dump of the resource name. The two leftmost columns of the dump show its contents as hexadecimal values, the least significant byte being represented by the rightmost two digits. The rightmost column represents its contents as ASCII text, the least significant byte being represented by the leftmost character.
Status	Status of the lock, information used internally by the lock manager.
Length	Length of the resource name.
Mode	Processor access mode of the namespace in which the resource block (RSB) associated with the lock resides.
Owner	Owner of the resource. Certain resources owned by the operating system list "System" as the owner. Resources owned by a group have the number (in octal) of the owning group in this field.
Copy	Indication of whether the lock is mastered on the local system or is a process copy.

[1]This display element is produced only by the SHOW PROCESS/LOCKS command.

Example

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

Lock id:  01000001   PID:     00000000   Flags:   NOQUEUE SYNCSTS SYSTEM
Par. id:  00000000   SUBLCKs:        0            CVTSYS
LKB:      80C9FD40   BLKAST:  00000000
PRIORTY:      0000

Granted at      EX   00000000-FFFFFFFF

Resource:      5F535953 24535953    SYS$SYS_  Status:  NOQUOTA
 Length   16   00000000 FF854449    ID......
 Exec. mode    00000000 00000000    ........
 System        00000000 00000000    ........

Local copy
```

```
Lock database
-------------

Lock id:  05000002   PID:     00000000   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  0100000E   SUBLCKs:        0            CVTSYS
LKB:      80CD0D40   BLKAST:  00000000
PRIORTY:      0000

Granted at      NL   00000000-FFFFFFFF

Resource:      09C27324 42313146    F11B$sÂ.  Status:  NOQUOTA
 Length   10   00000000 00000000    ........
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........

Process copy of lock 010002C0 on system 00010016 (FLAMS)

Lock database
-------------

Lock id:  02000003   PID:     00000000   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  00000000   SUBLCKs:        0            NOQUOTA CVTSYS
LKB:      80D317C0   BLKAST:  00000000
PRIORTY:      0000

Granted at      CR   00000000-FFFFFFFF

Resource:      4153445F 24535953    SYS$_DSA  Status:  NOQUOTA
 Length   10   00000000 00003A32    2:......
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........

Process copy of lock 0D000304 on system 00010014 (ROMRDR)


   .
   .
   .
SDA>  SHOW RESOURCE/LOCK=280009
Resource database
-----------------
Address of RSB:  80D31D00  GGMODE:       NL  Status: VALID
Parent RSB:      00000000  CGMODE:       NL
Sub-RSB count:           0  FGMODE:       NL
Lock Count:              1  CSID:   00010014 (ROMRDR)
BLKAST count:            0  RQSEQNM:    0000

Resource:      4153445F 24535953   SYS$_DSA  Valblk: 00000000 00000019
 Length   10   00000000 00003A32   2:......          00000000 00000000
 Kernel mode   00000000 00000000   ........
 System        00000000 00000000   ........  Seqnum: 00000011

Granted queue (Lock ID / Gr mode / Range):
 02000003  CR 00000000-FFFFFFFF

Conversion queue (Lock ID / Gr mode / Range -> Rq mode / Range):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode / Range):
     *** EMPTY QUEUE ***
```

This SDA session shows the output of the SHOW LOCK command for several locks. The SHOW RESOURCE command, executed for the last displayed lock, verifies that the lock is in the resource's granted queue. (See Table SDA–28 for a full explanation of the contents of the display of the SHOW RESOURCE command.)

SHOW MACHINE_CHECK

Displays the contents of the stored machine check frame. This command is valid for the DEC 4000 Alpha, DEC 7000 Alpha, and DEC 10000 Alpha computers only.

Format

SHOW MACHINE_CHECK [/FULL] [cpu-id]

Parameter

cpu-id
Numeric value from 00 to $1F_{16}$ indicating the identity of the processor for which context information is to be displayed. This parameter changes the SDA current CPU (the default) to the CPU specified with **cpu-id**. If you specify a value outside this range, or you specify the **cpu-id** of a processor that was not active at the time of the system failure, SDA displays the following message:

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

If you use the **cpu-id** parameter, the SHOW MACHINE_CHECK command performs an implicit SET CPU command, making the processor indicated by **cpu-id** the current CPU for subsequent SDA commands. (See the description of the SET CPU command and Section 4 for information on how this can affect the CPU context—and process context—in which SDA commands execute.)

Qualifier

/FULL
Specifies that a detailed version of the machine check information be displayed. This is currently identical to the default summary display.

Description

The SHOW MACHINE_CHECK command displays the contents of the stored machine check frame. A separate frame is allocated at boot time for every CPU in a multiple-CPU system. This command is valid for the DEC 4000 Alpha, DEC 7000 Alpha, and DEC 10000 Alpha computers only.

If no qualifier is specified, a summary version of the machine check frame is displayed.

The default **cpu-id** is the SDA current CPU.

Examples

1.
```
SDA> SHOW MACHINE_CHECK
CPU 00 Stored Machine Check Crash Data
--------------------------------------

Processor specific information:
-------------------------------
Exception address:    FFFFFFFF.800B0250    Exception Summary:  00000000.00000000
Pal base address:     00000000.00008000    Exception Mask:     00000000.00000000
HW Interrupt Request: 00000000.00000342    HW Interrupt Ena:   00000001.FFC01CE0
MM_CSR                00000000.00003640    ICCSR:              00000002.381F0000
D-cache address:      00000007.FFFFFFFF    D-cache status:     00000000.000002E0
BIU status:           00000000.00000050    BIU address [7..0]: 00000000.000060E0
BIU control:          00000008.50006447    Fill Address:       00000000.00006120
Single-bit syndrome:  00000000.00000000    Processor mchck VA: 00000000.00006190
A-box control:        00000000.0000040E    B-cache TAG:        00106100.83008828

System specific information:
----------------------------
Garbage bus info:     00200009 00000038    Device type:                000B8001
LCNR:                          00000001    Memory error:               00000000
LBER:                          00000009    Bus error synd 0,1: 00000000 00000000
Bus error cmd:        00048858 00AB1C88    Bus error synd 2,3: 00000000 0000002C
LEP mode:                      00010010    LEP lock address:           00041108
```

   The SHOW MACHINE_CHECK command in this SDA display shows the contents of the stored machine check frame.

2.
```
SDA> SHOW MACHINE_CHECK 1

CPU 01 Stored Machine Check Crash Data
--------------------------------------

Processor specific information:
-------------------------------
Exception address:    FFFFFFFF.800868A0    Exception Summary:  00000000.00000000
Pal base address:     00000000.00008000    Exception Mask:     00000000.00000000
HW Interrupt Request: 00000000.00000342    HW Interrupt Ena:   00000000.1FFE1CE0
MM_CSR                00000000.00005BF1    ICCSR:              00000000.081F0000
D-cache address:      00000007.FFFFFFFF    D-cache status:     00000000.000002E0
BIU status:           00000000.00000050    BIU address [7..0]: 00000000.000063E0
BIU control:          00000008.50006447    Fill Address:       00000000.00006420
Single-bit syndrome:  00000000.00000000    Processor mchck VA: 00000000.00006490
A-box control:        00000000.0000040E    B-cache TAG:        35028EA0.50833828

System specific information:
----------------------------
Garbage bus info:     00210001 00000038    Device type:                000B8001
LCNR:                          00000001    Memory error:               00000080
LBER:                          00040209    Bus error synd 0,1: 00000000 00000000
Bus error cmd:        00048858 00AB1C88    Bus error synd 2,3: 00000000 0000002C
LEP mode:                      00010010    LEP lock address:           00041108
```

   The SHOW MACHINE_CHECK command in this SDA display shows the contents of the stored machine check frame for **cpu-id** 01.

SHOW PAGE_TABLE

Displays a range of system page table entries, the entire system page table, or the entire global page table.

Format

SHOW PAGE_TABLE {range|/FREE|/GLOBAL|/GPT|/PT
|/S0S1 (d)|/SPTW|/ALL|option}
{/L1|/L2|/L3 (d)}

Parameter

range
Range of virtual addresses for which SDA is to display page table entries. You can express a range using the following syntax:

m Displays the single page table entry tht corresponds to virtual address *m*

m:n Displays the page table entries that correspond to the range of virtual addresses from *m* to *n*

m;n Displays the page table entries that correspond to a range of *n* bytes starting at virtual address *m*

Qualifiers

/FREE
Causes the free starting addresses of blocks of free page table entries in the specified range to be displayed.

/GLOBAL
Lists the global page table.

/GPT
Specifies the portion of page table space that maps the global page table as the address range.

/L1
Lists the L1 page table entries for the portion of memory specified.

/L2
Lists the L2 page table entries for the portion of memory specified.

/L3
Lists the L3 page table entries for the portion of memory specified. This qualifier is the default level.

/PT
Specifies page table space as the address range as viewed from system context.

/S0S1
Specifies S0 and S1 space as the address range. The default portion of memory.

/S2
Specifies S2 space as the address range.

/SPTW
Displays the contents of the system page table window.

/ALL
Displays the equivalent to all of /S0S1, /S2, /SPTW, /PT, /GPT, and /GLOBAL.

Option

= ALL
Displays with the SHOW PAGE = All command the page table entries for all shared (system) addresses, without regard to the section of memory being referenced. This option can be qualified only by one of the /L1, /L2, or /L3 qualifiers.

Note

The /L1, /L2, and /L3 qualifiers are ignored when use with the /FREE, /GLOBAL, and /SPTW qualifiers.

Description

For each virtual address displayed by the SHOW PAGE_TABLE command, the first eight columns of the listing provide the associated page table entry and describe its location, characteristics, and contents. SDA obtains this information from the system page table. Table SDA–13 describes the information displayed by the SHOW PAGE_TABLE command.

Table SDA–13 Virtual Page Information in the SHOW PAGE_TABLE Display

Value	Meaning
MAPPED ADDRESS	Virtual address that marks the base of the virtual page.
PTE ADDRESS	Virtual address of the page table entry that maps the virtual page.
PTE	Contents of the page table entry, a quadword that describes a system virtual page.
TYPE	Type of virtual page. Table SDA–14 shows the eight types and their meanings.
READ	A code, derived from bits in the PTE, that designates the processor access modes (kernel, executive, supervisor, or user) for which read access is granted.
WRIT	A code, derived from bits in the PTE, that designates the processor access modes (kernel, executive, supervisor, or user) for which write access is granted.
BITS	Letters that represent the setting of a bit or a combination of bits in the PTE. These bits indicate attributes of a page. Table SDA–15 shows the codes and their meanings.
GH	Contents of granularity hint bits.

Table SDA–14 Type of Virtual Pages

Type	Meaning
VALID	Valid page (in main memory)
TRANS	Transitional page (between main memory and page lists)
DZERO	Demand-allocated, zero-filled page
PGFIL	Page within a paging file
STX	Section table's index page
GPTX	Index page for a global page table
IOPAG	Page in I/O address space
NXMEM	Page not represented in physical memory. The page frame number (PFN) of this page is not mapped by any of the system's memory controllers. This indicates an error condition.

Table SDA–15 Bits In the PTE

Code	Meaning
A	Address space match is set.
M	Page has been modified.
L	Page is locked into a working set.
K	Owner can access the page in kernel mode.
E	Owner can access the page in executive mode.
S	Owner can access the page in supervisor mode.
U	Owner can access the page in user mode.

If the virtual page has been mapped to a physical page, the last six columns of the listing include information from the page frame number (PFN) database Otherwise, the section is left blank. Table SDA–16 describes the physical page information displayed by the SHOW PAGE_TABLE command.

Table SDA–16 Physical Page Information in the SHOW PAGE_TABLE Display

Category	Meaning
PGTYP	Type of physical page. Table SDA–17 shows the types of physical page.
LOC	Location of the page within the system. Table SDA–18 shows the 10 types with their meaning.
BAK	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	Number of references being made to this page.
FLINK	Forward link within PFN database that points to the next physical page; this longword also acts as the count of the number of processes that are sharing this global section.
BLINK	Backward link within PFN database; also acts as an index into the working set list.

Table SDA–17 Types of Physical Pages

Page Type	Meaning
PROCESS	Page is part of process space.
SYSTEM	Page is part of system space.
GLOBAL	Page is part of a global section.
PPGTBL	Page is part of a process page table.
PHD[1]	Page is part of a process PHD.
PPT(Ln)[1]	Page is a process page table page at level *n*.
SPT(Ln)[1]	Page is a system page table page at level *n*.
GPGTBL	Page is part of a global page table.
GBLWRT	Page is part of a global, writable section.
SHPT[2]	Page is part of a shared page table.
UNKNOWN	Unknown.

[1]These page types are variants of the PPGTBL page type.

[2]The SHPT page type is a variant of the GBLWRT page type.

Table SDA–18 Location of the Page

Location	Meaning
ACTIVE	Page is in a working set.
MFYLST	Page is in the modified-page list.
FRELST	Page is in the free-page list.
BADLST	Page is in the bad-page list.
RELPND	Release of the page is pending.
RDERR	Page has had an error during an attempted read operation.
PAGOUT	Page is being written into a paging file.
PAGIN	Page is being brought into memory from a paging file.
ZROLST	Page is in the zeroed-page list.
UNKNWN	Page is in unknown list.

SDA indicates pages are inaccessible by displaying one of the following messages:

```
-------   1 null page:       VA   FFFFFFFE.00064000     PTE   FFFFFFFD.FF800190
------- 974 null pages:      VA   FFFFFFFE.00064000     PTE   FFFFFFFD.FF800190
                            -to- FFFFFFFE.007FE000     -to-  FFFFFFFD.FF801FF8
```

In this case, the page table entries are not in use (page referenced is inaccessible)

```
-------      1 entry not in memory:    VA   FFFFFFFE.00800000   PTE   FFFFFFFD.FF802000
------- 784384 entries not in memory:  VA   FFFFFFFE.00800000   PTE   FFFFFFFD.FF802000
                                       -to- FFFFFFFF.7F7FE000   -to-  FFFFFFFD.FFDFDFF8
```

In this case, the page table entries to not exist (PTE itself is enaccessible)

```
-------    1 free PTE:    VA   FFFFFFFF.7F800000   PTE   FFFFFFFD.FFDFE000
------- 1000 free PTEs:   VA   FFFFFFFF.7F800000   PTE   FFFFFFFD.FFDFE000
                          -to- FFFFFFFF.7FFCE000   -to-  FFFFFFFD.FFDFFF38
```

In this case, the page table entries are in the list of free system pages

In each case, "VA" is the MAPPED ADDRESS of the skipped entry, and "PTE" is the PTE ADDRESS of the skipped entry.

SHOW PFN_DATA

Displays information that is contained in the page lists and PFN database.

Format

SHOW_PFN_DATA {[/qualifier] | pfn [{:end-pfn | ;length}]}

Parameters

pfn
Page frame number (PFN) of the physical page for which information is to be displayed.

length
Specifies the length of the PFN list to be displayed. When you specify the **length** parameter, a range of PFNs is displayed. This range starts at the PFN specified by the **pfn** parameter and contains the number of entries specified by the **length** parameter.

end-pfn
Specifies the last PFN to be displayed. When you specify the **end-pfn** parameter, a range of PFNs is displayed. This range starts at the PFN specified by the **pfn** parameter and ends with the PFN specified by the **end-pfn** parameter.

Qualifiers

/ADDRESS=<PFN-entry-address>
Displays the PFN database entry at the address specified. The address specified is rounded to the nearest entry address so if you have an address that points to one of the fields of the entry, the correct database entry will still be found.

/ALL
Displays the free-page list, modified-page list, and bad-page list. This is the default behavior of the SHOW PFN_DATA command. SDA precedes each list with a count of the pages it contains and its low and high limits.

/BAD
Displays the bad-page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/COLOR [={*n* | ALL}]
Displays data on page coloring. Table SDA–19 shows the command options available with this qualifier.

Table SDA–19 Command Options with the /COLOR Qualifier

Options	Meaning
/COLOR with no value	Displays a summary of the lengths of the colored page lists for both free pages and zeroed pages.
/COLOR=*n* where *n* is a color number	Displays the data in the PFN lists (for the specified color) for both free and zeroed pages.
/COLOR=ALL	Displays the data in the PFN lists (for all colors), for both free and zeroed free pages.
/COLOR=*n* or /COLOR=ALL with /FREE or /ZERO	Displays only the data in the PFN list (for the specified color or all colors), for either free or zeroed free pages as appropriate. The qualifiers /BAD and /MODIFIED are ignored with /COLOR=*n* and /COLOR=ALL.
/COLOR without an option specified together with one or more of /FREE, /ZERO, /BAD, or /MODIFIED	Displays the color summary in addition to the display of the requested list(s).

For more information on page coloring, see *OpenVMS System Management Utilities Reference Manual: M–Z*.

/FREE
Displays the free-page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/MODIFIED
Displays the modified-page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/SYSTEM
Displays the entire PFN database in order by page frame number, starting at PFN 0000.

/ZERO
Displays the contents of the zeroed free page list.

Description

For each page frame number it displays, the SHOW PFN_DATA command lists information used in translating physical page addresses to virtual page addresses. The display has two lines of information. Table SDA–20 shows the first line's fields; Table SDA–21 shows the second line's fields.

Table SDA–20 Page Frame Number Information—Line One Fields

Item	Contents
PFN	Page frame number.
DB ADDRESS	Address of PFN structure for this page.

(continued on next page)

Table SDA–20 (Cont.) Page Frame Number Information—Line One Fields

Item	Contents
PT PFN	PFN of the page page table page that maps this page.
BAK	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.
FLINK	Forward link within PFN database that points to the next physical page; this longword also acts as the count of the number of processes that are sharing this global section.
BLINK	Backward link within PFN database; also acts as an index into the working set list.
SWP/BO	Either a swap file page number or a buffer object reference count, depending on a flag set in the page state field.
LOC	Location of the page within the system. Table SDA–18 shows the location with their 10 types and meaning.
FLAGS	Displays in text form the flags that are set in page state. Table SDA–22 shows the possible flags and their meaning.

Table SDA–21 Page Frame Number Information—Line Two Fields

Item	Contents
Blank	
PTE ADDRESS	System virtual address of the page table entry that describes the virtual page mapped into this physical page. If no virtual page is mapped into this physical page then "<no backpointer>" is displayed.
Blank	
Blank	
Blank	
Blank	
REFCNT	Number of references being made to this page.
PAGETYP	Type of physical page. See Table SDA–17 for the types of physical pages and their meanings.
FLAGS	If the page is a page table page, then the contents of the PRN$W_PT_VAL_CNT, PFN$W_PT_LCK_CNT, and PFN$W_PT_WIN_CNT fields are displayed. The format is as follows: VALCNT = *nnnn* LCKCNT = *nnnn* WINCNT = *nnnn*

Table SDA–22 Flags Set in Page State

Flag	Meaning
BUFOBJ	Set if any buffer objects reference this page.
COLLISION	Empty collision queue when page read is complete.
BADPAG	Bad page.
RPTEVT	Report event on I/O completion.
DELCON	Delete PFN when REFCNT=*0*.
MODIFY	Dirty page (modified).
UNAVAILABLE	PFN is unavailable. Most likely a console page.

SHOW POOL

Displays the contents of the nonpaged dynamic storage pool and the paged dynamic storage pool. You can display part or all of each pool. If no range or qualifiers are specified, the default is SHOW POOL/ALL. Optionally, it displays the nonpaged pool history ring buffer.

Format

SHOW POOL {{range | /ALL (d) | /BAP | /NONPAGED | /PAGED}
[/FREE | /HEADER | /SUMMARY | /TYPE=*block-type* |
/SUBTYPE=*block-type*] | /RING_BUFFER | /STATISTICS
[{/NONPAGED | /BAP}]}

Parameter

range
Range of virtual addresses in pool that SDA is to examine. You can express a range using the following syntax:

m:n Range of virtual addresses in pool from *m* to *n*

m;n Range of virtual addresses in pool starting at *m* and continuing for *n* bytes

Qualifiers

/ALL
Displays the entire contents of memory, except for those portions of memory that are free (available). This is the default behavior of the SHOW POOL command.

/BAP
Displays the contents of the bus-addressable dynamic storage pool currently in use.

/FREE
Displays the entire contents, both allocated and free, of the specified region or regions of pool. Use the /FREE qualifier with a **range** to show all of the used and free pool in the given range.

/HEADER
Displays only the first 16 longwords of each data block found within the specified region or regions of pool.

/NONPAGED
Displays the contents of the nonpaged dynamic storage pool currently in use.

/PAGED
Displays the contents of the paged dynamic storage pool currently in use.

/RING_BUFFER
Displays the contents of the nonpaged pool history ring buffer if pool checking has been enabled. Entries are displayed in reverse chronological order; that is, most to least recent. This qualifier is mutually exclusive of all other SHOW POOL qualifiers.

/STATISTICS
Displays usage statistics about each lookaside list. For each list, its queue header address, packet size, attempts, fails, and deallocations are displayed. This can be further qualified by using /NONPAGED, or /BAP to only display statistics for a specified pool area.

/SUBTYPE=*block-type*
Displays the blocks within the specified region or regions of pool that are of the indicated **block-type**. If SDA finds no blocks of that subtype in the pool region, it displays a blank screen, followed by an allocation summary of the region. For information on block-type, see block-type in the Description section.

/SUMMARY
Displays *only* an allocation summary for each specified region of pool.

/TYPE=*block-type*
Displays the blocks within the specified region or regions of pool that are of the indicated **block-type**. If SDA finds no blocks of that type in the pool region, it displays a blank screen, followed by an allocation summary of the region. For information on block-type, see block-type in the Description section.

Note

Some qualifiers cannot be used in the same command as some other qualifiers. Regard the first group of qualifiers (/FREE, etc) as filter qualifiers, the second group of qualifiers (/range, etc) as range specifying qualifiers, and the third group as additional exclusive qualifiers.

Description

The SHOW POOL command displays information about the contents of any specified region of pool in an 8-column format. The contents of the full display, from left to right, are listed as follows:

Column 1 contains the type of control block that starts at the virtual address in pool indicated in column 2. If SDA cannot interpret the block type, it displays a block type of "UNKNOWN." Column 3 lists the number of bytes (in decimal) of memory allocated to the block.

The remaining columns contain a dump of the contents of the block, in 4-longword intervals, until the block is complete. Columns 4 through 7 display, from right to left, the contents in hexadecimal; column 8 displays, from left to right, the contents in ASCII. If the ASCII value of a byte is not a printing character, SDA displays a period (.) instead.

For each region of pool it examines, the SHOW POOL command displays an allocation summary. The summary displays the range of addresses used by this region of pool, the address of the header for the free list for this region of pool, and, where applicable, the address of the array of headers for the lookaside lists for this region of pool. Following this is a 4-column table which lists, in column 2, the types of control block identified in the region and records the number of each in column 1. The last two columns represent the amount of the pool region occupied by each type of control block: column 3 records the total number of bytes, and column 4 records the percentage. The summary concludes with an indication of the number of bytes used within the particular pool region, as well

as the number of bytes remaining. It provides an estimate of the percentage of the region that has been allocated.

Block-type

Each block of pool has a type field (a byte containing a value in the range of 0-255). Many of these type values have names associated that are defined in $DYNDEF in SYS$LIBRARY:LIB.MLB. The block-type specified in the /TYPE qualifier of the SHOW POOL command can either be the value of the pool type or its associated name.

Some pool block-types have an additional subtype field (also a byte containing a value in the range of 0-255), many of which also have names associated. The block-type specified in the /SUBTYPE qualifier of the SHOW POOL command can either be the value of the pool type or its associated name. However, if given as a value, a /TYPE qualifier (giving a value or name) must also be specified. Note also that /TYPE and /SUBTYPE are interchangeable if the block-type is given by name. Table SDA–23 shows several examples.

Table SDA–23 /TYPE and /SUBTYPE Qualifier Examples

/TYPE and /SUBTYPE Qualifiers	Meaning
/TYPE = *CI*	All CI blocks regardless of subtype
/TYPE = *CI_MSG*	All CI blocks with subtype CI_MSG
/TYPE = *MISC/SUBTYPE = 120*	All MISC blocks with subtype 120
/TYPE = *0/SUBTYPE = 0*	All blocks with TYPE and SUBTYPE both zero

Examples

1.
```
SDA> SHOW POOL G0BADE00;260
Non-paged dynamic storage pool
------------------------------
                    Dump of blocks allocated from non-paged pool


   CIMSG   FFFFFFFF.80BADE00   144
                                    001000DA 003C0090 0000A900 00036FF0 .o........<.....
                                    D9B3001C 00000000 A0B5001D 35E60017 ...5............
                                    41414141 00000600 65EA0004 00000600 .......e....AAAA
                                    41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                                    41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
      .
      .
      .
   UNKNOWN FFFFFFFF.80BADE90   112
                                    41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                                    41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                                    41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                                    41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
      .
      .
      .
   CIDG    FFFFFFFF.80BADED0   144
                                    807708BB 003B0090 0004D7E0 000008F0 .........;...w.
                                    61616161 61616161 61616161 016CE87C ..l.aaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
      .
      .
      .
   UNKNOWN FFFFFFFF.80BADF60    64
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
      .
      .
      .
   CIDG    FFFFFFFF.80BADFA0   144
                                    807708BB 003B0090 0003FFC0 0004B1B0 .........;...w.
                                    61616161 61616161 61616161 016CE94C L.l.aaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
      .
      .
      .
   UNKNOWN FFFFFFFF.80BAE030    48
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                                    61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa


         Start              End             Length
-----------------  -----------------  -----------------
FFFFFFFF.80D0E000  FFFFFFFF.80ECE000  00000000.001C0000

Free list header:                     FFFFFFFF.80C0593C
Lookaside list header array:          FFFFFFFF.80C50378
```

```
Summary of Non-Paged Pool contents
----------------------------------
    3  UNKNOWN   =      176  (29%)
    2  CIDG      =      288  (48%)
    1  CIMSG     =      144  (24%)

Total space used = 608 out of 608 total bytes, 0 bytes left

Total space utilization = 100%
```

This example examines 608 (260_{16}) bytes of nonpaged pool, starting at address $80BADE00_{16}$, which happens to be the starting address of the CIMSG block listed in the example's output. SDA attempts to identify allocated blocks as it proceeds through the specified region of pool, and displays an allocation summary when it completes the listing.

2.
```
SDA>  SHOW POOL/PAGED/HEADER
Paged dynamic storage pool
--------------------------
                    Dump of blocks allocated from paged pool


RSHT    FFFFFFFF.8024FE00   528
                                    802DC710 00380210 00000000 FFFFFF80 ..........8...-.
LNM     FFFFFFFF.80250010    96
                                    8015B847 00400060 802D75A0 00000000 .....u-.‘.@.G...
LNM     FFFFFFFF.80250070    48
                                    8015B847 01400030 802500A0 802D7400 .t-...%.0.@.G...
LNM     FFFFFFFF.802500A0    96
                                    8015B847 02400060 802DC170 80250070 p.%.p.-.‘.@.G...
LNM     FFFFFFFF.80250100    48
                                    8015B847 00400030 802DC510 802E1B60 ‘.....-.0.@.G...
   .
   .
   .
```

The SHOW POOL/PAGED/HEADER command displays only the name of each block allocated from paged pool, its starting address, its size, and the first 4 longwords of its contents.

SHOW PORTS

Displays those portions of the port descriptor table (PDT) that are port independent.

Format

SHOW PORTS [/qualifier[,...]]

Parameters

None.

Qualifiers

/ADDRESS=pdt-address
Displays the specified port descriptor table (PDT). You can find the **pdt-address** for any active connection on the system in the **PDT summary page** display of the SHOW PORTS command. This command also defines the symbol PE_PDT. The connection descriptor table (CDT) addresses are also stored in many individual data structures related to System Communications Services (SCS) connections; for instance, in the path block displays of the SHOW CLUSTER/SCS command.

/BUS=bus-address
Displays bus (LAN device) structure data.

/CHANNEL=channel-address
Displays channel (CH) data.

/DEVICE
Displays the network path description for a channel.

/MESSAGE
Displays the message data associated with a virtual circuit (VC).

/NODE=node
Shows only the virtual circuit block associated with the specific node. When you use the /NODE qualifier, you must also specify the address of the PDT using the /ADDRESS qualifier.

/VC=vc-address
Displays the virtual circuit data.

Description

The SHOW PORTS command provides port-independent information from the port descriptor table (PDT) for those CI ports with full System Communications Services (SCS) connections. This information is used by all SCS port drivers.

Note that the SHOW PORTS command does not display similar information about UDA ports, BDA ports, and similar controllers.

The SHOW PORTS command also defines symbols for PEDRIVER based on the cluster configuration. These symbols include the following information:

- Virtual circuit (VC) control blocks for each of the remote systems
- Bus data structure for each of the local LAN adapters
- Some of the data structures used by both PEDRIVER and the LAN drivers

The following symbols are defined automatically:

- VC_nodename—Example: VC_NODE1, address of the local node's virtual circuit to node NODE1.
- CH_nodename—The preferred channel for the virtual circuit. For example, CH_NODE1, address of the local node's preferred channel to node NODE1.
- BUS_busname—Example: BUS_ETA, address of the local node's bus structure associated with LAN adapter ETA0.
- PE_PDT—Address of PEDRIVER's port descriptor table.
- MGMT_VCRP_busname—Example: MGMT_VCRP_ETA, address of the management VCRP for bus ETA.
- HELLO_VCRP_busname—Example: HELLO_VCRP_ETA, address of the HELLO message VCRP for bus ETA.
- VCIB_busname—Example: VCIB_ETA, address of the VCIB for bus ETA.
- UCB_LAVC_busname—Example: UCB_LAVC_ETA, address of the LAN device's UCB used for the local-area OpenVMS Cluster protocol.
- UCB0_LAVC_busname—Example: UCB0_LAVC_ETA, address of the LAN device's template UCB.
- LDC_LAVC_busname—Example: LDC_LAVC_ETA, address of the LDC structure associated with LAN device ETA.
- LSB_LAVC_busname—Example: LSB_LAVC_ETA, address of the LSB structure associated with LAN device ETA.

These symbols equate to system addresses for the corresponding data structures. You can use these symbols, or an address, after the equal sign in SDA commands.

The SHOW PORTS command produces several displays. The initial display, the **PDT summary page**, lists the PDT address, port type, device name, and driver name for each PDT. Subsequent displays provide information taken from each PDT listed on the summary page.

You can use the /ADDRESS qualifier to the SHOW PORTS command to produce more detailed information about a specific port. The first display of the SHOW PORTS/ADDRESS command duplicates the last display of the SHOW PORTS command, listing information stored in the port's PDT. Subsequent displays list information about the port blocks and virtual circuits associated with the port.

Example

```
SDA> SHOW PORTS/ADDRESS=80618400

    --- Port Descriptor Table (PDT) 80618400 ---

Type: 03 pe
Characteristics: 0000



   --- Port Block 80618BC0 ---

Status: 0001 authorize
VC Count: 3
Secs Since Last Zeroed: 18635

SBUF Size             516     LBUF Size         1848     Next Refork    1863571
SBUF Count              9     LBUF Count           1     Forks Count     217383
SBUF Max              768     LBUF Max           384     Refork Count         0
SBUF Quo               11     LBUF Quo             1     SCS Messages    198478
SBUF Miss               9     LBUF Miss          249     VC Queue Cnt     12308
SBUF Allocs        205551     LBUF Allocs        598     TQE Received     18635
SBUFs In Use            0     LBUFs In Use         0     Timer Done       18635
Peak SBUF In Use        9     Peak LBUF In Use     2     RWAITQ Count       781
SBUF Queue Empty        0     LBUF Queue Empty     0     LDL Buf/Msg       6218
TR SBUF Queue Empty     0
No SBUF for ACK         0

Bus Addr  Bus     LAN Address    Error Count Last Error   Time of Last Error
--------  ---  ----------------- ----------- ---------- ----------------------
80619280  LCL  00-00-00-00-00-00           0
806198C0  ESA  AA-00-04-00-C7-FF           0


   --- Virtual Circuit (VC) Summary ---

VC Addr     Node    SCS ID  Lcl ID    Status Summary        Last Event Time
--------  --------  ------  ------  -----------------   ----------------------
8062A240  FLAM5      65479  223/DF  open,path           31-AUG-1995 17:30:17.05
8062BA40  VANDQ1     64894  222/DE  open,path           31-AUG-1995 17:30:18.87
8062BEC0  ROMRDR     64515  221/DD  open,path           31-AUG-1995 17:30:19.07
```

This example illustrates the output produced by the SHOW PORTS command for the PDT at address 80618400.

SHOW PROCESS

Displays the software and hardware context of any process in the balance set.

Format

SHOW PROCESS {[process-name] | ALL | /ADDRESS=*pcb_address* | /ID=*nn*
| /INDEX=nn | /SYSTEM}
[/ALL | /BUFFER_OBJECTS | /CHANNEL | /IMAGES | /LOCKS |
/PAGE_TABLES | /PCB | /PHD | /PROCESS_SECTION_TABLE
[/SECTION_INDEX=id] | RDE [=*id*] | /REGIONS [=*id*]
| /REGISTERS | /RMS [=option[,...]]
| /SEMAPHORE | /SYSTEM | /THREADS | /WORKING_SET_LIST]

Parameters

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

process-name
Name of the process for which information is to be displayed. Use of the **process-name** parameter, the /ADDRESS qualifier, the /INDEX qualifier, or the /SYSTEM qualifier causes the SHOW PROCESS command to perform an implicit SET PROCESS command, making the indicated process the current process for subsequent SDA commands. You can determine the names of the processes in the system by issuing a SHOW SUMMARY command.

The **process-name** can contain up to 15 letters and numerals, including the underscore (_) and dollar sign ($). If it contains any other characters, you must enclose the **process-name** in quotation marks (" ").

Qualifiers

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

/ALL
Displays all information shown by the following qualifiers:

/PCB
/PHD
/REGISTERS
/WORKING_SET_LIST
/PROCESS_SECTION_TABLE
/PAGE_TABLES
/CHANNEL
/BUFFER_OBJECTS
/IMAGES
/RMS

/BUFFER_OBJECTS
Displays all the buffer objects that a process has created.

/CHANNEL
Displays information about the I/O channels assigned to the process.

/IMAGES
Displays the address of the image control block, the start and end addresses of the image, the activation code, the protected and shareable flags, the image name, and the major and minor IDs of the image. The /IMAGES qualifier also displays the base, end, image offset, and section type for installed resident images in use by this process.

See the *OpenVMS Linker Utility Manual* and the Install utility chapter in the *OpenVMS System Management Utilities Reference Manual* for more information on images installed using the /RESIDENT qualifier.

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

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

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

/LOCKS
Displays the lock management locks owned by the current process.

The /LOCKS qualifier produces a display similar in format to that produced by the SHOW LOCKS command. See Table SDA–12 for additional information.

/PAGE_TABLES {range|/P0 (d)|/P1|/P2|/PT| /RDE=*ID*|/REGIONS=*id*|=ALL} {/L1|/L2|/L3 (d)}
Displays the page tables of the process P0 (process), P1 (control), P2, or PT (page table) region, or, optionally, page table entries for a **range** of addresses. The page table entries at the level specified by /L1, /L2, or /L3 (the default) are displayed.

The /RDE=*id* or /REGIONS=*id* displays the page tables for the address range of the specified address region. When no ID is specified, the page tables are displayed for all the process-permanent and user-defined regions.

You can express a **range** using the following syntax:

m	Displays the single page table entry that corresponds to virtual address *m*
m:n	Displays the page table entries that correspond to the range of virtual addresses from *m* to *n*
m;n	Displays the page table entries that correspond to a range of *n* bytes, starting at virtual address *m*
=ALL	Displays the entire page table for the process from address zero to the end of process-private page table space by using /PAGE_TABLES=ALL.

/PCB
Displays the information contained in the process control block (PCB). This is the default behavior of the SHOW PROCESS command.

/PHD
Lists the information included in the process header (PHD).

/PROCESS_SECTION_TABLE [/SECTION_INDEX=*id*]
Lists the information contained in the process section table (PST). The /SECTION_INDEX=*id* qualifier used with /PROCESS_SECTION_TABLE displays the process section table entry for the specified section.

/RDE [=*id*
/REGIONS [=*id*]
Lists the information contained in the process region table for the specified region. If no region is specified, the entire table is displayed, including the process-permanent regions. The qualifiers /RDE [=*id*] and /REGIONS [=*id*] may be used interchangeably.

/REGISTERS
Lists the hardware context of the process, as reflected in the process registers stored in the hardware privileged context block (HWPCB), its kernel stack, and possibly, in its PHD.

/RMS[=*option*[,...]]
Displays certain specified RMS data structures for each image I/O or process permanent I/O file the process has open. To display RMS data structures for process-permanent files, specify the PIO option to this qualifier.

SDA determines the structures to be displayed according to either of the following methods:

- If you provide the name of a structure or structures in the **option** parameter, SHOW PROCESS/RMS displays information from only the specified structures. (See Table SDA–10 for a list of keywords that may be supplied as options.)
- If you do not specify an **option**, SHOW PROCESS/RMS displays the current list of options as shown by the SHOW RMS command and set by the SET RMS command.

/SEMAPHORE
Displays the Inner Mode Semaphore for a multithreaded process.

/SYSTEM
Displays the system process control block. Use of the **process-name** parameter, the /INDEX qualifier, or the /SYSTEM qualifier causes the SHOW PROCESS command to perform an implicit SET PROCESS command, making the indicated process the current process for subsequent SDA commands. (See the description of the SET PROCESS command and Section 4 for information on how this can affect the process context—and CPU context—in which SDA commands execute.) The system PCB and process header (PHD) parallel the data structures that describe processes. They contain the system working set, global section table, global page table, and other systemwide data.

/THREADS
Displays the software and hardware context of all the threads associated with the current process.

/WORKING_SET_LIST [={ PPT | PROCESS | LOCKED | GLOBAL | MODIFIED | *n*}]

Displays the contents of the requested entries of the working set list for the process. If no option is specified, then all working set list entries are displayed.

Table SDA–24 shows the options available with SHOW PROCESS/WORKING_SET_LIST.

Table SDA–24 Options for the /WORKING_SET_LIST Qualifier

Options	Results
PPT	Displays process page table pages.
PROCESS	Displays process private pages.
LOCKED	Displays pages locked into the process's working set.
GLOBAL	Displays global pages currently in the working set of the process.
MODIFIED	Displays working set list entries marked modified.
n	Displays a specific working set list entry, where *n* is the working set list index (WSLX) of the entry of interest.

Description

The SHOW PROCESS command displays information about the process specified by **process-name**, the process specified in the /INDEX qualifier, the system process, or all processes. The SHOW PROCESS command performs an implicit SET PROCESS command under certain uses of its qualifiers and parameters, as noted previously. By default, the SHOW PROCESS command produces information about the SDA current process, as defined in Section 4.

The default of the SHOW PROCESS command provides information taken from the software process control block (PCB). This is the first display provided by the /ALL qualifier and the only display provided by the /PCB qualifier. This information describes the following characteristics of the process:

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

Among the displayed information are the process PID, EPID, priority, job information block (JIB) address, and process header (PHD) address. SHOW PROCESS also describes the resources owned by the process, such as event flags and mutexes. The "State" field records the process current scheduling state; in a multiprocessing system, the display indicates the CPU ID of any process whose state is CUR.

The SHOW PROCESS/ALL command displays additional process-specific information, also provided by several of the individual qualifiers to the command.

The **process header** display, also produced by the /PHD qualifier, provides information taken from the PHD, which is swapped into memory when the process becomes part of the balance set. Each item listed in the display reflects a quantity, count, or limit for the process use of the following resources:

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

- I/O activity
- CPU activity

The **process registers** display, also produced by the /REGISTERS qualifier, describes the process hardware context, as reflected in its registers.

There are two places where a process hardware context is stored:

- If the process is currently executing on a processor in the Alpha system (that is, in the CUR scheduling state), its hardware context is contained in that processor's registers. (That is, the process registers and the processor's registers contain identical values, as illustrated by a SHOW CPU command for that processor or a SHOW CRASH command if the process was current at the time of the system failure).
- If the process is not executing, its privileged hardware context is stored in the part of the PHD known as the HWPCB. Its integer register context is stored on its kernel stack. Its floating-point registers are stored in its PHD.

The **process registers** display first lists those registers stored in the HWPCB, kernel stack, and PHD ("Saved process registers"). If the process to be displayed is currently executing on a processor in the Alpha system, the display then lists the processor's registers ("Active registers for the current process"). In each section, the display lists the registers in the following groups:

- Integer registers (R0 through R29)
- Special-purpose registers (PC and PS)
- Stack pointers (KSP, ESP, SSP, and USP)
- Page table base register (PTBR)
- AST enable and summary registers (ASTEN and ASTSR)
- Address space number register (ASN)

The **working set information** and **working set list** displays, also produced by the /WORKING_SET_LIST qualifier, describe those virtual pages that the process can access without a page fault. After a brief description of the size, scope, and characteristics of the working set list itself, SDA displays information for each entry in the working set list as shown in Table SDA–25.

Table SDA–25 Working Set List Entry Information in the SHOW PROCESS Display

Column	Contents
INDEX	Index into the working set list at which information for this entry can be found
ADDRESS	Virtual address of the page that this entry describes

(continued on next page)

Table SDA–25 (Cont.) Working Set List Entry Information in the SHOW PROCESS Display

Column	Contents
STATUS	Three columns that list the following status information: • Page status of VALID • Type of physical page (See Table SDA–17) • Indication of whether the page is locked into the working set

When SDA locates one or more unused working set entries, or entries that do not match the specified option, it issues the following message:

```
--- n entries not displayed
```

In this message, *n* is the number (in decimal) of contiguous entries not displayed.

The **process section table information** and **process section table** displays, also produced by the /PROCESS_SECTION_TABLE qualifier, list each entry in the process section table (PST) and display the offsets to, and the indices of, the first free entry and last used entry.

SDA displays the information listed in Table SDA–26 for each PST entry.

Table SDA–26 Process Section Table Entry Information in the SHOW PROCESS Display

Part	Definition
INDEX	Index number of the entry. Entries in the process section table begin at the highest location in the table, and the table expands toward lower addresses.
ADDRESS	Address of the process section table entry.
SECTION ADDRESS	Virtual address that marks the beginning of the first page of the section described by this entry.
PAGELETS	Length of the process section. This is in units of pagelets, except for a PFN-mapped section in which the units are pages.
WINDOW	Address of the window control block on which the section file is open.
VBN	Virtual block number. The number of the file's virtual block that is mapped into the section's first page.
CCB	Address of the channel control block on which the section file is open.
REFCNT	Number of pages of this section that are currently mapped.
FLINK	Forward link. The pointer to the next entry in the PST list.
BLINK	Backward link. The pointer to the previous entry in the PST list.
FLAGS	Flags that describe the access that processes have to the process section.

The **P0 page table**, **P1 page table**, and **P2 page table** displays, also produced by the /PAGE_TABLES qualifier, display listings of the process page table entries

in the same format as that produced by the SHOW PAGE_TABLE command (see Tables SDA–13 through Table SDA–18.)

The **process active channels** display, the last produced by SHOW PROCESS /ALL and the only one produced by the /CHANNEL qualifier, displays the following information for each I/O channel assigned to the process:

Column	Contents
Channel	Number of the channel
Window	Address of the window control block (WCB) for the file if the device is a file-oriented device; zero otherwise
Status	Status of the device: "Busy" if the device has an I/O operation outstanding; blank otherwise
Device/file accessed	Name of the device and, if applicable, name of the file being accessed on that device

The information listed under the heading "Device/file accessed" varies from channel to channel and from process to process. SDA displays certain information according to the conditions listed in Table SDA–27.

Table SDA–27 Process I/O Channel Information in the SHOW PROCESS Display

Information Displayed[1]	Type of Process
dcuu:	SDA displays this information for devices that are not file structured, such as terminals, and for processes that do not open files in the normal way.
dcuu:*filespec*	SDA displays this information only if you are examining a running system, and only if your process has enough privilege to translate the *file-id* into the *filespec*.
dcuu:(*file-id*)*filespec*	SDA displays this information only when you are examining a dump. The *filespec* corresponds to the *file-id* on the device listed. If you are examining a dump from your own system, the *filespec* is probably valid. If you are examining a dump from another system, the *filespec* is probably meaningless in the context of your system.
dcuu:(*file-id*)	The *file-id* no longer points to a valid *filespec*, as when you look at a dump from another system; or the process in which you are running SDA does not have enough privilege to translate the *file-id* into the corresponding *filespec*.
section file	Indicates that the file in question is mapped into the processes' memory.

[1]This table uses the following conventions to identify the information displayed:
dcuu:(*file-id*)*filespec*
where:
dcuu: is the name of the device.
file-id is the RMS file identification.
filespec is the full file specification, including directory name.

Examples

1. SDA> SHOW PROCESS

```
Process index: 001A   Name: VERIFICATION   Extended PID: 0000051A
------------------------------------------------------------------

Process status:  22040023   RES,PHDRES,INTER
        status2: 00000001   QUANTUM_RESCHED

PCB address              80613240    JIB address              805B1B40
PHD address              80C3A000    Swapfile disk address    00000000
KTB vector address       80D775AC    HWPCB address            81260080
Callback vector address  00000000    Termination mailbox          0000
Master internal PID      0005001A    Subprocess count                0
Creator extended PID     00000000    Creator internal PID     00000000
Previous CPU Id          00000000    Current CPU Id           00000000
Previous ASNSEQ  0000000000000001    Previous ASN     000000000000002E
Initial process priority        4    Delete pending count            0
# open files allowed left     100    Direct I/O count/limit        150/150
UIC                [00001,000004]    Buffered I/O count/limit      149/150
Abs time of last event   005D9941    BUFIO byte count/limit      32128/32128
ASTs remaining                247    # of threads                    1
Swapped copy of LEFC0    00000000    Timer entries allowed left     20
Swapped copy of LEFC1    00000000    Active page table count         0
Global cluster 2 pointer 00000000    Process WS page count         250
Global cluster 3 pointer 00000000    Global WS page count            0

Extended PID: 00000052     Thread index: 0000
--------------------------------------------
Current capabilities:    System: 0000000C  QUORUM,RUN
                         User:   00000000

Permanent capabilities:  System: 0000000C  QUORUM,RUN
                         User:   00000000
Current affinities:      00000000
Permanent affinities:    00000000
Thread status:           02040001
       status2:          00000001

KTB address              80D772C0    HWPCB address            81260080
PKTA address             7FFEFFC0    Callback vector address  00000000
Internal PID             00010012    Callback error           00000000
Extended PID             00000052    Current CPU id           00000000
State                       LEF      Flags                    00000000
Base priority                 4      Current priority                9
Waiting EF cluster            0      Event flag wait mask     DFFFFFFF
CPU since last quantum       FFF1    Mutex count                     0
ASTs active                  NONE
```

The SHOW PROCESS command displays information taken from the software PCB of VERIFICATION, the SDA current process. According to the "State" field in the display, process VERIFICATION is current.

2. SDA> SHOW PROCESS/ALL

```
Process index: 001A   Name: VERIFICATION   Extended PID: 0000051A
------------------------------------------------------------------

Process status:  22040023   RES,PHDRES,INTER
        status2: 00000001   QUANTUM_RESCHED
```

```
PCB address              80613240    JIB address              805B1B40
PHD address              80C3A000    Swapfile disk address    00000000
KTB vector address       80D775AC    HWPCB address            81260080
Callback vector address  00000000    Termination mailbox          0000
Master internal PID      0005001A    Subprocess count                0
Creator extended PID     00000000    Creator internal PID     00000000
Previous CPU Id          00000000    Current CPU Id           00000000
Previous ASNSEQ  0000000000000001    Previous ASN     000000000000002E
Initial process priority        4    Delete pending count            0
# open files allowed left     100    Direct I/O count/limit        150/150
UIC                [00001,000004]    Buffered I/O count/limit      149/150
Abs time of last event   005D9941    BUFIO byte count/limit      32128/32128
ASTs remaining                247    # of threads                    1
Swapped copy of LEFC0    00000000    Timer entries allowed left     20
Swapped copy of LEFC1    00000000    Active page table count         0
Global cluster 2 pointer 00000000    Process WS page count         250
Global cluster 3 pointer 00000000    Global WS page count            0

Extended PID: 00000052     Thread index: 0000
--------------------------------------------
Current capabilities:    System: 0000000C  QUORUM,RUN
                         User:   00000000

Permanent capabilities:  System: 0000000C  QUORUM,RUN
                         User:   00000000
Current affinities:      00000000
Permanent affinities:    00000000
Thread status:           02040001
       status2:          00000001

KTB address              80D772C0    HWPCB address            81260080
PKTA address             7FFEFFC0    Callback vector address  00000000
Internal PID             00010012    Callback error           00000000
Extended PID             00000052    Current CPU id           00000000
State                       LEF      Flags                    00000000
Base priority                 4      Current priority                9
Waiting EF cluster            0      Event flag wait mask     DFFFFFFF
CPU since last quantum      FFF1     Mutex count                     0
ASTs active                 NONE


Saved process registers
-----------------------
R0   = 00000000.00000001  R1   = 00000000.00000000  R2   = FFFFFFFF.80C8FEB0
R3   = 00000000.7FFCF680  R4   = 00000000.0000001D  R5   = 00000000.7FFCF680
R6   = 00000000.7FFCE4C0  R7   = 00000000.7FFAC9F0  R8   = 00000000.7B015EB8
R9   = 00000000.7FFAC410  R10  = 00000000.7FFAD238  R11  = 00000000.7FFCE3E0
R12  = 00000000.00000000  R13  = FFFFFFFF.80C68AC0  R14  = 00000000.00000000
R15  = 00000000.7B0A17A0  R16  = FFFFFFFF.80C05F18  R17  = FFFFFFFF.80D772C0
R18  = 00000000.00000002  R19  = 00000000.00000001  R20  = 00000000.7FFF0010
R21  = FFFFFFFD.FF7FE000  R22  = FFFFFFFF.800CCFC8  R23  = 00000000.7FFA1FC0
R24  = 00000000.7B015EB8  R25  = 00000000.00000005  R26  = 00000000.00000FD2
R27  = FFFFFFFF.80C652A0  R28  = 00000000.7B0A17A0  FP   = 00000000.7FFAC280
PC   = FFFFFFFF.800CCFC8  PS   = 00000000.00000012
KSP  = 00000000.7FFA1EF0  ESP  = 00000000.7FFA6000  SSP  = 00000000.7FFAC270
USP  = 00000000.7B013AF0  PTBR = 00000000.00000552
AST{SR/EN}    = 0000000F  ASN  = 00000000.0000002E

Extended PID: 00000052     Thread index: 0000
--------------------------------------------
Process header
--------------
```

```
First free P0 VA  00000000.00000000    Accumulated CPU time       00000014
First free P1 VA  00000000.7B012000    Subprocess quota                10
First free P2 VA  00000000.80000000    ASTs enabled                  KESU
Free page file pages           3027    ASN sequence #     0000000000000001
Page fault cluster size           4    AST limit                      250
Page table cluster size           1    Process header index          0001
Flags                      00000084    Backup address vector      0005AFE8
Direct I/O count                 27    PTs having locked WSLEs           2
Buffered I/O count               86    PTs having valid WSLEs            4
Limit on CPU time          00000000    Active page tables                4
Maximum page file count        3125    Maximum active PTs                3
Total page faults               262    Guaranteed fluid WS pages        20
File limit                      100    Extra dynamic WS entries         94
Timer queue limit                20    Current page file template 00000000
Local event flag cluster 0 C0000001    Local event flag cluster 1 80000000
Page Table Base Register   00000552    Virtual PT Base   FFFFFFFC.00000000

Process page file assignments
-----------------------------

 PROCIDX  SYSIDX    REFCNT

    0         3         40 Current assignment
    1         0          0
    2         0          0
    3         0          0

Remaining reserved pages      20    Total reserved pages          20

Extended PID: 00000052     Thread index: 0000
--------------------------------------------
Working set information
-----------------------

First WSL entry          00000001   Current authorized working set size    250
First locked entry       00000007   Default (initial) working set size     125
First dynamic entry      00000009   Maximum working set allowed (quota)    250
Last entry replaced      00000079
Last entry in list       000000D3

Working set list
----------------

      INDEX        ADDRESS                STATUS

  00000001  FFFFFFFD.FF7FC000     VALID PPT(L1) WSLOCK
  00000002  FFFFFFFD.FF000000     VALID PPT(L2) WSLOCK
  00000003  FFFFFFFC.001FE000     VALID PPT(L3) WSLOCK
  00000004  00000000.7FFA0000     VALID PROCESS MODIFIED WSLOCK
  00000005  00000000.7FFF0000     VALID PROCESS WSLOCK
  00000006  FFFFFFFF.81260000     VALID PHD     WSLOCK
```

```
Locked entries:
  00000007  00000000.7B108000     VALID PROCESS WSLOCK
  00000008  00000000 7B10A000     VALID PROCESS WSLOCK
Dynamic entries:
  00000009  00000000.7B054000     VALID GLOBAL
  0000000A  00000000.7B0B0000     VALID GLOBAL
  0000000B  FFFFFFFC.001EC000     VALID PPT(L3) WSLOCK
  0000000C  00000000.7B0D0000     VALID GLOBAL
  0000000D  00000000.7B0C4000     VALID GLOBAL
  0000000E  00000000.7B0C0000     VALID GLOBAL
  0000000F  00000000.7FFA4000     VALID PROCESS
  00000010  00000000.7FFD0000     VALID PROCESS
  00000011  00000000.7FF96000     VALID PROCESS
  00000012  00000000.7B0C6000     VALID GLOBAL
  00000013  00000000.7B0DC000     VALID GLOBAL
  00000014  00000000.7B0E4000     VALID GLOBAL
  00000015  00000000.7B0E6000     VALID GLOBAL
  00000016  00000000.7B0DE000     VALID GLOBAL
  00000017  00000000.7FFAA000     VALID PROCESS
  00000018  00000000.7B0E2000     VALID GLOBAL
  00000019  00000000.7FFCE000     VALID PROCESS
  0000001A  00000000.7B0D2000     VALID GLOBAL
  0000001B  00000000.7B13E000     VALID PROCESS
  0000001C  00000000.7B140000     VALID PROCESS
  0000001D  00000000.7B0EA000     VALID GLOBAL
  0000001E  00000000.7B0CE000     VALID GLOBAL
  0000001F  00000000.7B068000     VALID GLOBAL
  00000020  00000000.7B0CC000     VALID GLOBAL
  00000021  00000000.7B07C000     VALID GLOBAL
  00000022  00000000.7B07E000     VALID GLOBAL
  00000023  00000000.7B084000     VALID GLOBAL
  00000024  00000000.7B086000     VALID GLOBAL
  00000025  00000000.7FFB8000     VALID PROCESS
  00000026  00000000.7B144000     VALID PROCESS
  00000027  FFFFFFFC.00000000     VALID PPT(L3)
  00000028  00000000.7FF88000     VALID PROCESS
  00000029  00000000.7FFBA000     VALID PROCESS

  ----  8 entries not displayed

  00000032  00000000.7FF8A000     VALID PROCESS

  ----  6 entries not displayed

  00000039  00000000.7B0D6000     VALID GLOBAL
  0000003A  00000000.7B0D8000     VALID GLOBAL

  ----  3 entries not displayed

  0000003E  00000000.7B0DA000     VALID GLOBAL

  ----  8 entries not displayed

  00000047  00000000.7B066000     VALID GLOBAL
  00000048  00000000.7B104000     VALID PROCESS
  00000049  00000000.7B0B8000     VALID GLOBAL
  0000004A  00000000.7B07A000     VALID GLOBAL

  ----  11 entries not displayed

  00000056  00000000.7B13A000     VALID PROCESS
  00000057  00000000.7B13C000     VALID PROCESS
```

```
      ----  81 entries not displayed

      000000A9  00000000.7FFEE000     VALID PROCESS
      000000AA  00000000.7B142000     VALID PROCESS
         000000AB  00000000.7FFB0000     VALID PROCESS
      000000AC  00000000.7B0FE000     VALID PROCESS
      000000AD  00000000.7B09E000     VALID PROCESS
      000000AE  00000000.7B0A0000     VALID PROCESS
      000000AF  00000000.7B0A2000     VALID PROCESS
      000000B0  00000000.7B0A4000     VALID PROCESS
      000000B1  00000000.7B100000     VALID PROCESS

      ----  18 entries not displayed

      000000C4  00000000.7B138000     VALID PROCESS


Process section table
---------------------

INDEX  ADDRESS    SECTION ADDRESS    PAGELETS   WINDOW     VBN       CCB      REFCNT  FLINK BLINK   FLAGS

0001   815D5FD8  00000000.00010000   00000001  80D234C0  00000003  7FF96020  00000000  0003  0002   CRF WRT AMOD=KRNL
0002   815D5FB0  00000000.00030000   00000001  80D234C0  00000004  7FF96020  00000001  0001  0003   AMOD=KRNL
0003   815D5F88  00000000.00040000   00000001  80D234C0  00000005  7FF96020  00000000  0002  0001   CRF WRT AMOD=KRNL

P0 Space
--------

 MAPPED ADDRESS      PTE ADDRESS           PTE        TYPE  READ WRIT BITS GH PGTYP   LOC          BAK       REFCNT  FLINK    BLINK

        --------            8 null pages:             VA   00000000.00000000          PTE  FFFFFFFC.00000000
                                                      -to- 00000000.0000E000          -to- FFFFFFFC.00000038

00000000.00010000 FFFFFFFC.00000040 000003E7.00160F09 VALID KESU NONE M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000034

        --------            7 null pages:             VA   00000000.00012000          PTE  FFFFFFFC.00000048
                                                      -to- 00000000.0001E000          -to- FFFFFFFC.00000078

00000000.00020000 FFFFFFFC.00000080 0000046E.0016FF09 VALID KESU KESU M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000037

        --------            7 null pages:             VA   00000000.00022000          PTE  FFFFFFFC.00000088
                                                      -to- 00000000.0002E000          -to- FFFFFFFC.000000B8

00000000.00030000 FFFFFFFC.000000C0 0000015C.00060F01 VALID KESU NONE --U- 0 PROCESS ACTIVE 00000002.00090000 0001 00000000 00000036

        --------            7 null pages:             VA   00000000.00032000          PTE  FFFFFFFC.000000C8
                                                      -to- 00000000.0003E000          -to- FFFFFFFC.000000F8

00000000.00040000 FFFFFFFC.00000100 0000014D.00163F09 VALID KESU KE-- M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000032

        --------          991 null pages:             VA   00000000.00042000          PTE  FFFFFFFC.00000108
                                                      -to- 00000000.007FE000          -to- FFFFFFFC.00001FF8


        --------       130048 entries not in memory:  VA   00000000.00800000          PTE  FFFFFFFC.00002000
                                                      -to- 00000000.3FFFE000          -to- FFFFFFFC.000FFFF8


P1 Space
--------

 MAPPED ADDRESS      PTE ADDRESS           PTE        TYPE  READ WRIT BITS GH PGTYP   LOC          BAK       REFCNT  FLINK    BLINK

        --------       119808 entries not in memory:  VA   00000000.40000000          PTE  FFFFFFFC.00100000
                                                      -to- 00000000.7A7FE000          -to- FFFFFFFC.001E9FF8


        --------         1020 null pages:             VA   00000000.7A800000          PTE  FFFFFFFC.001EA000
                                                      -to- 00000000.7AFF6000          -to- FFFFFFFC.001EBFD8

00000000.7AFF8000 FFFFFFFC.001EBFE0 00000000.0006FF00 DZERO KESU KESU --U- 0
00000000.7AFFA000 FFFFFFFC.001EBFE8 000003B4.0016FF09 VALID KESU KESU M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000033
00000000.7AFFC000 FFFFFFFC.001EBFF0 00001F3C.00147709 VALID KES- KES- M-S- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000029

...

00000000.7FFEE000 FFFFFFFC.001FFFB8 00000D68.0010FF09 VALID KESU KESU M-K- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000035
00000000.7FFF0000 FFFFFFFC.001FFFC0 00001EA3.10103F09 VALID KESU KE-- MLK- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000005

        --------            7 null pages:             VA   00000000.7FFF2000          PTE  FFFFFFFC.001FFFC8
                                                      -to- 00000000.7FFFE000          -to- FFFFFFFC.001FFFF8
```

ZK-8863A-GE

```
P2 Space
--------

 MAPPED ADDRESS      PTE ADDRESS           PTE        TYPE  READ WRIT BITS GH PGTYP   LOC          BAK       REFCNT  FLINK    BLINK

        --------   1071382528 entries not in memory:   VA   00000000.80000000          PTE  FFFFFFFC.00200000
                                                       -to- FFFFFFFB.FFFFE000          -to- FFFFFFFD.FEFFFFF8


PT Space
--------

 MAPPED ADDRESS      PTE ADDRESS           PTE        TYPE  READ WRIT BITS GH PGTYP   LOC          BAK       REFCNT  FLINK    BLINK

FFFFFFFC.00000000 FFFFFFFD.FF000000 00000DD2.40101309 VALID KE-- K--- M-K- 0 PPT(L3) ACTIVE 03000000.00000000 0001 00000004 00000019

        --------          244 null pages:              VA   FFFFFFFC.00002000          PTE  FFFFFFFD.FF000008
                                                       -to- FFFFFFFC.001E8000          -to- FFFFFFFD.FF0007A0

FFFFFFFC.001EA000 FFFFFFFD.FF0007A8 00001AB0.40101309 VALID KE-- K--- M-K- 0 PPT(L3) ACTIVE 03000000.00000000 0001 00000003 000000CB
FFFFFFFC.001EC000 FFFFFFFD.FF0007B0 0000182E.40101309 VALID KE-- K--- M-K- 0 PPT(L3) ACTIVE 03000000.00000000 0001 00000031 0000005B

        --------            8 null pages:              VA   FFFFFFFC.001EE000          PTE  FFFFFFFD.FF0007B8
                                                       -to- FFFFFFFC.001FC000          -to- FFFFFFFD.FF0007F0

FFFFFFFC.001FE000 FFFFFFFD.FF0007F8 00000CE9.40001309 VALID KE-- K--- -LK- 0 PPT(L3) ACTIVE 03000000.00000000 0001 00000014 00000003

        --------          768 null pages:              VA   FFFFFFFC.00200000          PTE  FFFFFFFD.FF000800
                                                       -to- FFFFFFFC.007FE000          -to- FFFFFFFD.FF001FF8


        --------      1045504 entries not in memory:   VA   FFFFFFFC.00800000          PTE  FFFFFFFD.FF002000
                                                       -to- FFFFFFFD.FEFFE000          -to- FFFFFFFD.FF7FBFF8

FFFFFFFD.FF000000 FFFFFFFD.FF7FC000 0000134D.40001109 VALID K--- K--- --K- 0 PPT(L2) ACTIVE 03000000.00000000 0001 00000004 00000002

        --------         1021 null pages:              VA   FFFFFFFD.FF002000          PTE  FFFFFFFD.FF7FC008
                                                       -to- FFFFFFFD.FF7FA000          -to- FFFFFFFD.FF7FDFE8

FFFFFFFD.FF7FC000 FFFFFFFD.FF7FDFF0 00000F6B.40001109 VALID K--- K--- --K- 0 PPT(L1) ACTIVE 00000000.815D4000 0001 00000001 00000001

ZK-8865A-GE
```

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

Channel  Window    Status Device/file accessed
-------  ------    ------ --------------------
  0010 00000000           DKB400:
  0040 00000000     Busy  OPA0:
  0060 00000000           OPA0:
  0090 80D83BC0           DKB400:(390,17,0)(section file)
  00A0 80D8AF40           DKB400:(3888,39,0)(section file)

                        Process activated images
                        ------------------------

  IMCB     Start     End      Sym Vect    Type   Image Name  Major ID, Minor ID
-------- -------- -------- -------- --------- ------------------------------

7FF88480 00010000 000401FF 00000000 MAIN         X 0,0
7FF8A4A0 80C03378 80C04E08 80C03378 GLBL         SYS$PUBLIC_VECTORS 93,1959106

Total images = 2               Pages allocated = 24

                    Process Buffered Objects
                    ------------------------

ADDRESS  ACMODE SEQUENCE  REFCNT    PID     PAGCNT   BASE PVA   BASE SVA
-------- ------ -------- -------- -------- -------- ---------- -----  --
No buffer objects for this proces
```

The SHOW PROCESS/ALL command displays information taken from the PCB of process VERIFICATION, and then proceeds to display the process header, the process registers, the process section table, the page tables of the process, images activated, and information about the I/O channels owned by the process. These displays may also be obtained by the /PCB, /PHD, /REGISTERS, /RDE,

/PROCESS_SECTION_TABLE, /P0, /P1, /P2, /PT, /IMAGES, and /CHANNEL qualifiers, respectively.

3. SDA> SHOW PROCESS/PAGE_TABLES/ADDRESS=805E7980

```
    P0 page table
    -------------

    MAPPED ADDRESS      PTE ADDRESS           PTE        TYPE  READ WRIT BITS GH PGTYP   LOC          BAK       REFCNT  FLINK    BLINK

           --------            8 null pages:             VA   00000000.00000000          PTE  FFFFFFFC.00000000
                                                         -to- 00000000.0000E000          -to- FFFFFFFC.00000038

   00000000.00010000 FFFFFFFC.00000040 000003E7.00160F09 VALID KESU NONE M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000034

           --------            7 null pages:             VA   00000000.00012000          PTE  FFFFFFFC.00000048
                                                         -to- 00000000.0001E000          -to- FFFFFFFC.00000078

   00000000.00020000 FFFFFFFC.00000080 0000046E.0016FF09 VALID KESU KESU M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000037

           --------            7 null pages:             VA   00000000.00022000          PTE  FFFFFFFC.00000088
                                                         -to- 00000000.0002E000          -to- FFFFFFFC.000000B8

   00000000.00030000 FFFFFFFC.000000C0 0000015C.00060F01 VALID KESU NONE --U- 0 PROCESS ACTIVE 00000002.00090000 0001 00000000 00000036

           --------            7 null pages:             VA   00000000.00032000          PTE  FFFFFFFC.000000C8
                                                         -to- 00000000.0003E000          -to- FFFFFFFC.000000F8

   00000000.00040000 FFFFFFFC.00000100 0000014D.00163F09 VALID KESU KE-- M-U- 0 PROCESS ACTIVE 03000000.00000000 0001 00000000 00000032

           --------          991 null pages:             VA   00000000.00042000          PTE  FFFFFFFC.00000108
                                                         -to- 00000000.007FE000          -to- FFFFFFFC.00001FF8


           --------       130048 entries not in memory:  VA   00000000.00800000          PTE  FFFFFFFC.00002000
                                                         -to- 00000000.3FFFE000          -to- FFFFFFFC.000FFFF8
```

ZK-8864A-GE

This example displays the page tables of a process whose PCB address is 805E7980.

4. SDA>SHOW PROCESS/BUFFER_OBJECTS

```
                        Process Buffered Objects
                        ------------------------

ADDRESS  ACMODE SEQUENCE  REFCNT    PID     PAGCNT      BASE PVA      BASE SVA
-------- ------ -------- -------- -------- -------- ----------------- --------
805E4580 User   00000008 00000001 00010020 00000001 00000000.00020000 826BC000
805E7880 User   00000009 00000001 00010020 00000001 00000000.00020000 826BE000
8057AEC0 User   0000000A 00000001 00010020 00000001 00000000.00020000 826C0000
805E6EC0 User   0000000B 00000001 00010020 00000001 00000000.00020000 82764000
```

The SHOW PROCESS/BUFFER_OBJECTS command displays all the buffered objects that a process has created.

5. SDA>SHOW PROCESS/IMAGES

```
                            Process activated images
                            ------------------------
```

```
  IMCB    Start     End    Sym Vect    Type      Image Name  Major ID,Minor ID
-------- -------- -------- -------- ------------ ---------------------------
7FF78810 00010000 001107FF 00000000 MAIN         SDA 0,0
7FF789B0 001E6000 002263FF 001E80B0 GLBL     SHR LBRSHR 2,9
7FF76480 001A4000 001E43FF 001A4950 GLBL     SHR SCRSHR 1,2900
7FF785A0 00112000 001A27FF 00186AE0 GLBL     SHR SMGSHR 1,104
7FF78060 7FC06000 7FC67FFF 7FC144B0 GLBL     SHR LIBRTL 1,1
           Base      End   ImageOff     Section Type
         80400000 80481C00 00000000   System Resident Code
         7FC06000 7FC16800 00090000   Shareable Address Data
         7FC26000 7FC27000 000B0000   Read-Write Data
         7FC36000 7FC3F600 000C0000   Shareable Read-Only Data
         7FC46000 7FC46200 000D0000   Read-Write Data
         7FC56000 7FC57000 000E0000   Demand Zero Data
         7FC66000 7FC67400 000F0000   Read-Write Data
7FF78330 7FC76000 7FCA7FFF 7FC86000 GLBL     SHR LIBOTS 1,3
           Base      End   ImageOff     Section Type
         80482000 8048FA00 00020000   System Resident Code
         7FC76000 7FC78600 00000000   Shareable Read-Only Data
         7FC86000 7FC87C00 00010000   Shareable Address Data
         7FCA6000 7FCA6200 00030000   Read-Write Data
7FF78130 80810110 8081C770 80810110 GLBL         SYS$BASE_IMAGE 114,15303694
7FF784D0 80802A18 80803FF8 80802A18 GLBL         SYS$PUBLIC_VECTORS 114,15295276

Total images = 8               Pages allocated = 344
```

The SHOW PROCESS/IMAGES command displays the address of the image control block; the start and end addresses of the image; the activation code; the protected and shareable flags; the image name; the major and minor IDs of the image; and the base, end, image offset, and section type for installed resident images.

SHOW RESOURCE

Displays information about all resources in the system, or about a resource associated with a specific lock.

Format

SHOW RESOURCE {/ADDRESS=*n*|/ALL (d)|/CACHED
| /LOCKID=*lock-id*|/NAME=*resource-name*}

Parameters

None.

Qualifiers

/ADDRESS=*n*
Displays information from the resource block at the specified address.

/ALL
Displays information from all resource blocks (RSBs) in the system. This is the default behavior of the SHOW RESOURCE command.

/CACHED
Displays resource blocks that are no longer valid. The memory for these resources is kept around so that later requests for resources can use them.

/LOCKID=*lock-id*
Displays information on the resource associated with the lock with the specified *lock-id*.

/NAME=*resource-name*
Displays information about a specific resource.

Description

The SHOW RESOURCE command displays the information listed in Table SDA–28 for each resource in the system or for the specific resource associated with the specified **lock-id**.

Table SDA–28 Resource Information in the SHOW RESOURCE Display

Field	Contents
Address of RSB	Address of the resource block (RSB) that describes this resource.
Parent RSB	Address of the RSB that is the parent of this RSB. This field is 00000000 if the RSB itself is a parent block.
Sub-RSB count	Number of RSBs of which this RSB is the parent. This field is 0 if the RSB has no sub-RSBs.
Lock Count	The total count of all locks on the resource.

(continued on next page)

Table SDA–28 (Cont.) Resource Information in the SHOW RESOURCE Display

Field	Contents
BLKAST count	Number of locks on this resource that have requested a blocking AST.
GGMODE	Indication of the most restrictive mode in which a lock on this resource has been granted. Table SDA–29 shows the fields and values and their meanings. They are shown in order from the least restrictive mode to the most restrictive. For information on conflicting and incompatible lock modes, see the *OpenVMS System Services Reference Manual*.
CGMODE	Indication of the most restrictive lock mode to which a lock on this resource is waiting to be converted. This does not include the mode for which the lock at the head of the conversion queue is waiting.
FGMODE	Indication of the full-range grant mode.
CSID	Cluster system identification number (CSID) and name of the node that owns the resource.
RQSEQNM	Sequence number of the request.
Status	The contents of the resource block status field.
Resource	Dump of the name of this resource, as stored at the end of the RSB. The first two columns are the hexadecimal representation of the name, with the least significant byte represented by the rightmost two digits in the rightmost column. The third column contains the ASCII representation of the name, the least significant byte being represented by the leftmost character in the column. Periods in this column represent values that correspond to nonprinting ASCII characters.
Valblk	Hexadecimal dump of the 16-byte block value block associated with this resource.
Seqnum	Sequence number associated with the resource's value block. If the number indicates that the value block is not valid, the words "Not valid" appear to the right of the number.
Granted queue	List of locks on this resource that have been granted. For each lock in the list, SDA displays the number of the lock and the lock mode in which the lock was granted.
Conversion queue	List of locks waiting to be converted from one mode to another. For each lock in the list, SDA displays the number of the lock, the mode in which the lock was granted, and the mode to which the lock is to be converted.

(continued on next page)

Table SDA–28 (Cont.) Resource Information in the SHOW RESOURCE Display

Field	Contents
Waiting queue	List of locks waiting to be granted. For each lock in the list, SDA displays the number of the lock and the mode requested for that lock.
Length	Length in bytes of the resource name.
Mode	Processor mode of the namespace in which this RSB resides.
Owner	Owner of the resource. Certain resources, owned by the operating system, list "System" as the owner. Locks owned by a group have the number (in octal) of the owning group in this field.

Table SDA–29 Lock on Resources

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

Example

```
SDA>  SHOW RESOURCE
Resource database
-----------------
Address of RSB:  80D93D80  GGMODE:       NL  Status: VALID
Parent RSB:      80D73980  CGMODE:       NL
Sub-RSB count:          0  FGMODE:       NL
Lock Count:             1  CSID:   00000000  (MYNODE)
BLKAST count:           0  RQSEQNM:    0000

Resource:      1C477324 42313146   F11B$sG.  Valblk: 00000001 00000001
 Length   10   00000000 00000000   ........          00000000 00000000
 Kernel mode   00000000 00000000   ........
 System        00000000 00000000   ........  Seqnum: 00001304

Granted queue (Lock ID / Gr mode / Range):
 50000076  NL 00000000-FFFFFFFF

Conversion queue (Lock ID / Gr mode / Range -> Rq mode / Range):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode / Range):
     *** EMPTY QUEUE ***


Resource database
-----------------
Address of RSB:  80D990C0  GGMODE:       NL  Status: VALID
Parent RSB:      80D73980  CGMODE:       NL
Sub-RSB count:          0  FGMODE:       NL
Lock Count:             1  CSID:   00000000  (MYNODE)
BLKAST count:           0  RQSEQNM:    0000
```

```
Resource:      1D357324 42313146   F11B$s5.  Valblk: 00000001 00000001
 Length   10   00000000 00000000   ........          00000000 00000000
 Kernel mode   00000000 00000000   ........
 System        00000000 00000000   ........  Seqnum: 00000002

Granted queue (Lock ID / Gr mode / Range):
 040006A3  NL 00000000-FFFFFFFF

Conversion queue (Lock ID / Gr mode / Range -> Rq mode / Range):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode / Range):
     *** EMPTY QUEUE ***

   .
   .
   .
```

The SHOW RESOURCE command displays information taken from the RSBs of all resources in the system. For instance, the RSB at $80D93D80_{16}$ is a parent block with no sub-RSBs.

SHOW RMD

Displays information contained in the reserved memory descriptors. Reserved memory is used within the system by memory-resident global sections.

Format

SHOW RMD [/QUALIFIERS]

Parameter

None

Qualifiers

/ADDRESS=*n*
Displays a specific reserved memory descriptor entry, given its address.

/ALL
Displays information in all the reserved memory descriptors. This qualifier is the default.

Description

The SHOW RMD displays information that resides in the reserved memory descriptors. Table SDA–30 shows the fields and their meaning.

Table SDA–30 RMD Fields

Field	Meaning
ADDRESS	Gives the address of the reserved memory descriptor.
NAME	Gives the name of the reserved memory descriptor.
FLAGS	Gives the settings of flags for specified reserved memory descriptor, as a hexadecimal number, then key flag bits are also displayed by name.
GROUP	Gives the UIC group that owns the reserved memory. This is given as -S- for system global reserved memory.
PFN	Gives starting page number of the reserved memory.
COUNT	Gives the number of pages reserved.
IN_USE	Gives the number of pages in use.
ZERO_PFN	Gives the next page number to be zeroed.

Example

```
SDA> SHOW RMD
Reserved Memory Descriptor List
-------------------------------

ADDRESS  NAME       GROUP   PFN     COUNT    IN_USE  ZERO_PFN  FLAGS

80D21200 MILORD2     -S-  00000000 00000100 00000000 00000000 00000000
80D21100 MILORD1     -S-  00000A00 00000080 00000000 00000A00 00000001 ALLOC
80D21280 MILORD2     -S-  00000000 00000001 00000000 00000000 00000040 PAGE_TABLES
80D21180 MILORD1     -S-  00000180 00000001 00000000 00000180 00000041 ALLOC PAGE_TABLES
```

SHOW RMS

Displays the RMS data structures selected by the SET RMS command to be included in the default display of the SHOW PROCESS/RMS command.

Format

SHOW RMS

Parameters

None.

Qualifiers

None.

Description

The SHOW RMS command lists the names of the data structures selected for the default display of the SHOW PROCESS/RMS command.

For a description of the significance of the options listed in the SHOW RMS display, see the description of the SET RMS command and Table SDA–10.

For an illustration of the information displayed by the SHOW PROCESS/RMS command, see the examples included in the description of the SHOW PROCESS command.

Examples

1.
```
SDA>  SHOW RMS

RMS Display Options:  IFB,IRB,IDX,BDB,BDBSUM,ASB,CCB,WCB,FCB,FAB,RAB,NAM,
XAB,RLB,BLB,BLBSUM,GBD,GBH,FWA,GBDSUM,JFB,NWA,RU,DRC,SFSB,GBSB
Display RMS structures for all IFI values.
```

   The SHOW RMS command displays the full set of options available for display by the SHOW PROCESS/RMS command. SDA, by default, selects the full set of RMS options at the beginning of an analysis.

2.
```
SDA>  SET RMS=(IFAB,CCB,WCB)
SDA>  SHOW RMS

RMS Display Options:  IFB,CCB,WCB
Display RMS structures for all IFI values.
```

   The SET RMS command establishes the IFB, CCB, and WCB as the structures to be displayed when the SHOW PROCESS/RMS command is issued. The SHOW RMS command verifies this selection of RMS options.

SHOW RSPID

Displays information about response IDs (RSPIDs) of all System Communications Services (SCS) connections or, optionally, a specific SCS connection.

Format

SHOW RSPID [/CONNECTION=cdt-address]

Parameters

None.

Qualifier

/CONNECTION=cdt-address
Displays RSPID information for the specific SCS connection whose connection descriptor table (CDT) address is provided in **cdt-address**. You can find the **cdt-address** for any active connection on the system in the **CDT summary page** display of the SHOW CONNECTIONS command. CDT addresses are also stored in many individual data structures related to SCS connections. These data structures include class driver request packets (CDRPs) and unit control blocks (UCBs) for class drivers that use SCS and cluster system blocks (CSBs) for the connection manager.

Description

Whenever a local system application (SYSAP) requires a response from a remote SYSAP, a unique number, called an RSPID, is assigned to the response by the local system. The RSPID is transmitted in the original request (as a means of identification), and the remote SYSAP returns the same RSPID in its response to the original request.

The SHOW RSPID command displays information taken from the response descriptor table (RDT), which lists the currently open local requests that require responses from SYSAPs at a remote node. For each RSPID, SDA displays the following information:

- RSPID value
- Address of the class driver request packet (CDRP), which generally represents the original request
- Address of the CDT that is using the RSPID
- Name of the local process using the RSPID
- Remote node from which a response is required (and has not yet been received)

Examples

1. SDA> SHOW RSPID

```
 --- Summary of Response Descriptor Table (RDT) 805E6F18 ---

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

39D00000     8062CC80        805E8710       VMS$VMScluster          VANDQ1
EE210001     80637260        805E8C90       VMS$DISK_CL_DRVR        ROMRDR
EE240002     806382E0        805E8DF0       VMS$DISK_CL_DRVR        VANDQ1
EE440003     806393E0        805E8F50       VMS$TAPE_CL_DRVR        VANDQ1
5DB90004     80636BC0        805E8870       VMS$VMScluster          ROMRDR
5C260005     80664040        805E8870       VMS$VMScluster          ROMRDR
38F80006     80664A80        805E8710       VMS$VMScluster          VANDQ1
```

This example shows the default output for the SHOW RSPID command.

2. SDA> SHOW RSPID/CONNECTION=805E8F50

```
 --- Summary of Response Descriptor Table (RDT) 805E6F18 ---

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

EE440003     806393E0        805E8F50       VMS$TAPE_CL_DRVR        VANDQ1
```

This example shows the output for a SHOW RSPID/CONNECTION command.

SHOW SPINLOCKS

Displays the multiprocessing synchronization data structures.

Format

SHOW SPINLOCKS {[name] | /ADDRESS=*expression* | /INDEX=*expression*}
[/OWNED | /DYNAMIC | /STATIC] [{/BRIEF | /FULL}]

Parameter

name
Name of the spin lock, fork lock, or device lock structure to be displayed. Device lock names are of the form [node$]lock, where node optionally indicates the OpenVMS Cluster node name (allocation class) and lock indicates the device and controller identification (for example, HAETAR$DUA).

Qualifiers

/ADDRESS=*expression*
Displays the lock at the address specified in **expression**. You can use the /ADDRESS qualifier to display a specific device lock; however, the name of the device lock is listed as "Unknown" in the display.

/BRIEF
Produces a condensed display of the lock information displayed by default by the SHOW SPINLOCKS command, including the following: address, spinlock name or device name, IPL or device IPL, rank, index, ownership depth, number of waiting CPUs, CPU ID of the owner CPU, and interlock status (depth of ownership).

/DYNAMIC
Displays information for all device locks in the system.

/FULL
Displays full descriptive and diagnostic information for each displayed spin lock, fork lock, or device lock.

/INDEX=*expression*
Displays the system spin lock whose index is specified in *expression*. You cannot use the /INDEX qualifier to display a device lock.

/OWNED
Displays information for all spin locks, fork locks, and device locks owned by the SDA current CPU. If a processor does not own any spin locks, SDA displays the following message:

```
No spinlocks currently owned by CPU xx
```

The *xx* represents the CPU ID of the processor.

/STATIC
Displays information for all system spin locks and fork locks.

Description

The SHOW SPINLOCKS command displays status and diagnostic information about the multiprocessing synchronization structures known as spin locks.

A **static spin lock** is a spin lock whose data structure is permanently assembled into the system. Static spin locks are accessed as indexes into a vector of longword addresses called the **spin lock vector**, the address of which is contained in SMP$AR_SPNLKVEC. System spin locks and fork locks are static spin locks. Table SDA–31 lists the static spin locks.

A **dynamic spin lock** is a spin lock that is created based on the configuration of a particular system. One such dynamic spin lock is the device lock SYSMAN creates when configuring a particular device. This device lock synchronizes access to the device's registers and certain UCB fields. The system creates a dynamic spin lock by allocating space from nonpaged pool, rather than assembling the lock into the system as it does in creating a static spin lock.

See the *Writing OpenVMS Alpha Device Drivers in C* for a full discussion of the role of spin locks in maintaining synchronization of kernel mode activities in a multiprocessing environment.

Table SDA–31 Static Spin Locks

Name	Description
QUEUEAST	Fork lock for queuing ASTs at IPL 6
FILSYS	Lock on file system structures
LCKMGR	Lock on all lock manager structures
IOLOCK8/SCS	Fork lock for executing a driver fork process at IPL 8
TX_SYNCH	Transaction processing lock
TIMER	Lock for adding and deleting timer queue entries and searching the timer queue
PORT	Template structure for dynamic spinlocks for ports with multiple devices
IO_MISC	Miscellaneous short term I/O locks
MMG	Lock on memory management, PFN database, swapper, modified page writer, and creation of per-CPU database structures
SCHED	Lock on process control blocks (PCBs), scheduler database, and mutex acquisition and release structures
IOLOCK9	Fork lock for executing a driver fork process at IPL 9
IOLOCK10	Fork lock for executing a driver fork process at IPL 10
IOLOCK11	Fork lock for executing a driver fork process at IPL 11
MAILBOX	Lock for sending messages to mailboxes
POOL	Lock on nonpaged pool database
PERFMON	Lock for I/O performance monitoring

(continued on next page)

Table SDA–31 (Cont.) Static Spin Locks

Name	Description
INVALIDATE	Lock for system space translation buffer (TB) invalidation
HWCLK	Lock on hardware clock database, including the quadword containing the due time of the first timer queue entry (EXE$GQ_1ST_TIME) and the quadword containing the system time (EXE$GQ_SYSTIME)
MEGA	Lock for serializing access to fork-wait queue
EMB/MCHECK	Lock for allocating and releasing error-logging buffers and synchronizing certain machine error handling

For each spin lock, fork lock, or device lock in the system, SHOW SPINLOCKS provides the following information:

- Name of the spin lock (or device name for the device lock)
- Address of the spinlock data structure (SPL)
- The owner CPU's CPU ID
- IPL at which allocation of the lock is synchronized on a local processor
- Number of nested acquisitions of the spin lock by the processor owning the spin lock ("Ownership Depth")
- Rank of the spin lock
- Number of processors waiting to obtain the spin lock
- Spinlock index (for static spin locks only)
- Timeout interval for spinlock acquisition (in terms of 10 milliseconds)

SHOW SPINLOCKS/BRIEF produces a condensed display of this same information.

If the system under analysis was executing with full-checking multiprocessing enabled (according to the setting of the MULTIPROCESSING system parameter), SHOW SPINLOCKS/FULL adds to the spinlock display the last eight PCs at which the lock was acquired or released. If applicable, SDA also displays the PC of the last release of multiple, nested acquisitions of the lock.

If no spin lock name, address, or index is given, then information is displayed for all applicable spin locks.

Examples

1.
```
SDA> SHOW SPINLOCKS
System static spinlock structures
---------------------------------
EMB                               Address   80424480
Owner CPU ID           None       DIPL      0000001F
Ownership Depth    00000000       Rank      00000000
CPUs Waiting       00000000       Index     00000020
Timeout Interval   000186A0

EMB                               Address   80424480
Owner CPU ID           None       DIPL      0000001F
Ownership Depth    00000000       Rank      00000000
CPUs Waiting       00000000       Index     00000020
Timeout Interval   000186A0

MEGA                              Address   80424500
Owner CPU ID           None       DIPL      00000016
Ownership Depth    00000000       Rank      00000002
CPUs Waiting       00000000       Index     00000022
Timeout Interval   000186A0

HWCLK                             Address   80424580
Owner CPU ID           None       DIPL      00000016
Ownership Depth    00000000       Rank      00000004
CPUs Waiting       00000000       Index     00000024
Timeout Interval   000186A0

   .
   .
   .
System dynamic spinlock structures
----------------------------------
OPA                               Address   8041E880
Owner CPU ID           None       DIPL      00000014
Ownership Depth    00000000       Rank      FFFFFFFF
CPUs Waiting       00000000
Timeout Interval   000186A0

MBA                               Address   80424780
Owner CPU ID           None       DIPL      0000000B
Ownership Depth    00000000       Rank      0000000C
CPUs Waiting       00000000       Index     0000002C
Timeout Interval   000186A0

NLA                               Address   80424780
Owner CPU ID           None       DIPL      0000000B
Ownership Depth    00000000       Rank      0000000C
CPUs Waiting       00000000       Index     0000002C
Timeout Interval   000186A0

PKI                               Address   80552800
Owner CPU ID           None       DIPL      00000014
Ownership Depth    00000000       Rank      FFFFFFFF
CPUs Waiting       00000000
Timeout Interval   000186A0

   .
   .
   .
```

This excerpt illustrates the default output of the SHOW SPINLOCKS command.

2.
```
SDA> SHOW SPINLOCKS/BRIEF
Address  Spnlck Name  IPL   Rank     Index    Depth  #Waiting Ownr CPU Interlock

------------------------------------------------------------------------------

8041F400 EMB         001F 00000000 00000020 00000000 00000000   None     Free
8041F400 EMB         001F 00000000 00000020 00000000 00000000   None     Free
8041F480 MEGA        001F 00000002 00000022 00000000 00000000   None     Free
8041F500 HWCLK       0016 00000004 00000024 00000000 00000000   None     Free
8041F580 INVALIDATE  0015 00000006 00000026 00000000 00000000   None     Free
8041F600 PERFMON     000F 00000008 00000028 00000000 00000000   None     Free
8041F680 POOL        000B 0000000A 0000002A 00000000 00000000   None     Free
8041F700 MAILBOX     000B 0000000C 0000002C 00000000 00000000   None     Free
8041F780 IOLOCK11    000B 0000000E 0000002E 00000000 00000000   None     Free
8041F800 IOLOCK10    000A 0000000F 0000002F 00000000 00000000   None     Free
8041F880 IOLOCK9     0009 00000010 00000030 00000000 00000000   None     Free
8041F900 SCHED       0008 00000012 00000032 00000000 00000000   None     Free
8041F980 MMG         0008 00000014 00000034 00000000 00000000   None     Free
8041FA00 IO_MISC     0008 00000016 00000036 00000000 00000000   None     Free
8041FA80 TIMER       0008 00000018 00000038 00000000 00000000   None     Free
8041FB00 TX_SYNCH    0008 00000019 00000039 00000000 00000000   None     Free
8041FB80 SCS         0008 0000001A 0000003A 00000000 00000000   None     Free
8041FC00 FILSYS      0008 0000001C 0000003C 00000000 00000000   None     Free
8041FC80 QUEUEAST    0006 0000001E 0000003E 00000000 00000000   None     Free
80419880 PIPERA$OPA  0015 FFFFFFFF          00000000 00000000   None     Free
8041F700 PIPERA$MBA  000B 0000000C 0000002C 00000000 00000000   None     Free
8041F700 PIPERA$NLA  000B 0000000C 0000002C 00000000 00000000   None     Free
805E9900 PIPERA$DKB  0016 FFFFFFFF          00000000 00000000   None     Free
805E9E80 PIPERA$PKB  0015 FFFFFFFF          00000000 00000000   None     Free
8041FB80 PIPERA$FTA  0008 0000001A 0000003A 00000000 00000000   None     Free
805B9400 PIPERA$PKA  0015 FFFFFFFF          00000000 00000000   None     Free
805BBC00 PIPERA$DKA  0016 FFFFFFFF          00000000 00000000   None     Free
805BC780 PIPERA$ESA  0015 FFFFFFFF          00000000 00000000   None     Free
805BE080 PIPERA$TTA  0015 FFFFFFFF          00000000 00000000   None     Free
805BEB00 PIPERA$SOA  0015 FFFFFFFF          00000000 00000000   None     Free
8041FB80 PIPERA$NET  0008 0000001A 0000003A 00000000 00000000   None     Free
8041FB80 PIPERA$NDA  0008 0000001A 0000003A 00000000 00000000   None     Free
8041FB80 PIPERA$RTA  0008 0000001A 0000003A 00000000 00000000   None     Free
8041FB80 PIPERA$RTB  0008 0000001A 0000003A 00000000 00000000   None     Free
8041FB80 PIPERA$LTA  0008 0000001A 0000003A 00000000 00000000   None     Free
8041FB80 PIPERA$RTC  0008 0000001A 0000003A 00000000 00000000   None     Free
8041FB80 PIPERA$PDA  0008 0000001A 0000003A 00000000 00000000   None     Free
```

This excerpt illustrates the condensed form of the display produced in the first example.

SHOW STACK

Displays the location and contents of the process stacks (of the SDA current process) and the system stack.

Format

SHOW STACK {range|/ALL|[/EXECUTIVE|/INTERRUPT|/KERNEL
|/SUPERVISOR|/SYSTEM|/USER]} {/LONG|/QUAD (d)}

Parameter

range
Range of memory locations you want to display in stack format. You can express a **range** using the following syntax:

m:n Range of virtual addresses from *m* to *n*

m;n Range of virtual addresses starting at *m* and continuing for *n* bytes

Qualifiers

/ALL
Displays the locations and contents of the four process stacks for the current SDA process and the system stack.

/EXECUTIVE
Shows the executive stack for the SDA current process.

/INTERRUPT
The interrupt stack does not exist in OpenVMS Alpha. This qualifier shows the system stack and is retained for compatibility with OpenVMS VAX.

/KERNEL
Shows the kernel stack for the SDA current process.

/LONG
Displays longword width stacks. If this qualifier is not specified, SDA by default displays quadword width stacks.

/QUAD
Displays quadword width stacks. This is the default.

/SUPERVISOR
Shows the supervisor stack for the SDA current process.

/SYSTEM
Shows the system stack.

/USER
Shows the user stack for the SDA current process.

Description

The SHOW STACK command, by default, displays the stack that was in use when the system failed, or, in the analysis of a running system, the current operating stack. For a process that became the SDA current process as the result of a SET PROCESS command, the SHOW STACK command by default shows its current operating stack.

The various qualifiers to the command can display any of the four per-process stacks for the SDA current process, as well as the system stack for the SDA current CPU.

You can define SDA process and CPU context by using the SET CPU, SHOW CPU, SHOW CRASH, SET PROCESS, and SHOW PROCESS commands as indicated in their command descriptions. A complete discussion of SDA context control appears in Section 4.

SDA provides the following information in each stack display:

Section	Contents
Identity of stack	SDA indicates whether the stack is a process stack (user, supervisor, executive, or kernel) or the system stack.
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 8 bytes (one quadword), unless you specify the /LONG qualifier in which case addresses are listed in increments of 4 (one longword).
Stack contents	SDA lists the contents of the stack in a column to the right of the stack addresses.
Symbols	SDA attempts to display the contents of a location symbolically, using a symbol and an offset. If the address cannot be symbolized, this column is left blank.
Canonical stack	When displaying the kernel stack of a non-current process in a crash dump, SDA identifies the stack locations used by the scheduler to store the register contents of the process.

If a stack is empty, the display shows the following:

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

Example

```
SDA> SHOW STACK
Current Operating Stack (SYSTEM):
                        FFFFFFFF.8244BD08  FFFFFFFF.800600FC  SCH$REPORT_EVENT_C+000FC
                        FFFFFFFF.8244BD10  00000000.00000002
                        FFFFFFFF.8244BD18  00000000.00000005
                        FFFFFFFF.8244BD20  FFFFFFFF.8060C7C0
                  SP => FFFFFFFF.8244BD28  FFFFFFFF.8244BEE8
                        FFFFFFFF.8244BD30  FFFFFFFF.80018960  EXE$HWCLKINT_C+00260
                        FFFFFFFF.8244BD38  00000000.000001B8
                        FFFFFFFF.8244BD40  00000000.00000050
                        FFFFFFFF.8244BD48  00000000.00000210  UCB$N_RSID+00002
                        FFFFFFFF.8244BD50  00000000.00000000
                        FFFFFFFF.8244BD58  00000000.00000000
                        FFFFFFFF.8244BD60  FFFFFFFF.804045D0  SCH$GQ_IDLE_CPUS
                        FFFFFFFF.8244BD68  FFFFFFFF.8041A340  EXE$GL_FKWAITFL+00020
                        FFFFFFFF.8244BD70  00000000.00000250  UCB$T_MSGDATA+00034
                        FFFFFFFF.8244BD78  00000000.00000001
CHF$IS_MCH_ARGS         FFFFFFFF.8244BD80  00000000.0000002B
CHF$PH_MCH_FRAME        FFFFFFFF.8244BD88  FFFFFFFF.8244BFB0
CHF$IS_MCH_DEPTH        FFFFFFFF.8244BD90  80000000.FFFFFFFD  G
CHF$PH_MCH_DADDR        FFFFFFFF.8244BD98  00000000.00001600  CTL$C_CLIDATASZ+00060
CHF$PH_MCH_ESF_ADDR     FFFFFFFF.8244BDA0  FFFFFFFF.8244BF40
CHF$PH_MCH_SIG_ADDR     FFFFFFFF.8244BDA8  FFFFFFFF.8244BEE8
CHF$IH_MCH_SAVR0        FFFFFFFF.8244BDB0  FFFFFFFF.8041FB00  SMP$RELEASEL+00640
CHF$IH_MCH_SAVR1        FFFFFFFF.8244BDB8  00000000.00000000
CHF$IH_MCH_SAVR16       FFFFFFFF.8244BDC0  00000000.0000000D
CHF$IH_MCH_SAVR17       FFFFFFFF.8244BDC8  0000FFF0.00007E04
CHF$IH_MCH_SAVR18       FFFFFFFF.8244BDD0  00000000.00000000
CHF$IH_MCH_SAVR19       FFFFFFFF.8244BDD8  00000000.00000001
CHF$IH_MCH_SAVR20       FFFFFFFF.8244BDE0  00000000.00000000
CHF$IH_MCH_SAVR21       FFFFFFFF.8244BDE8  FFFFFFFF.805AE4B6  SISR+0006E
CHF$IH_MCH_SAVR22       FFFFFFFF.8244BDF0  00000000.00000001
CHF$IH_MCH_SAVR23       FFFFFFFF.8244BDF8  00000000.00000010
CHF$IH_MCH_SAVR24       FFFFFFFF.8244BE00  00000000.00000008
CHF$IH_MCH_SAVR25       FFFFFFFF.8244BE08  00000000.00000010
CHF$IH_MCH_SAVR26       FFFFFFFF.8244BE10  00000000.00000001
CHF$IH_MCH_SAVR27       FFFFFFFF.8244BE18  00000000.00000000
CHF$IH_MCH_SAVR28       FFFFFFFF.8244BE20  FFFFFFFF.804045D0  SCH$GQ_IDLE_CPUS
                        FFFFFFFF.8244BE28  30000000.00000300  UCB$L_PI_SVA
                        FFFFFFFF.8244BE30  FFFFFFFF.80040F6C  EXE$REFLECT_C+00950
                        FFFFFFFF.8244BE38  18000000.00000300  UCB$L_PI_SVA
                        FFFFFFFF.8244BE40  FFFFFFFF.804267A0  EXE$CONTSIGNAL+00228
                        FFFFFFFF.8244BE48  00000000.7FFD00A8  PIO$GW_IIOIMPA
                        FFFFFFFF.8244BE50  00000003.00000000
                        FFFFFFFF.8244BE58  FFFFFFFF.8003FC20  EXE$CONNECT_SERVICES_C+00920
                        FFFFFFFF.8244BE60  FFFFFFFF.8041FB00  SMP$RELEASEL+00640
                        FFFFFFFF.8244BE68  00000000.00000000
                        FFFFFFFF.8244BE70  FFFFFFFF.8042CD50  SCH$WAIT_PROC+00060
                        FFFFFFFF.8244BE78  00000000.0000000D
                        FFFFFFFF.8244BE80  0000FFF0.00007E04
                        FFFFFFFF.8244BE88  00000000.00000000
                        FFFFFFFF.8244BE90  00000000.00000001
                        FFFFFFFF.8244BE98  00000000.00000000
                        FFFFFFFF.8244BEA0  FFFFFFFF.805AE4B6  SISR+0006E
                        FFFFFFFF.8244BEA8  00000000.00000001
                        FFFFFFFF.8244BEB0  00000000.00000010
                        FFFFFFFF.8244BEB8  00000000.00000008
                        FFFFFFFF.8244BEC0  00000000.00000010
                        FFFFFFFF.8244BEC8  00000000.00000001
                        FFFFFFFF.8244BED0  00000000.00000000
                        FFFFFFFF.8244BED8  FFFFFFFF.804045D0  SCH$GQ_IDLE_CPUS
                        FFFFFFFF.8244BEE0  00000000.00000001
```

```
CHF$L_SIG_ARGS          FFFFFFFF.8244BEE8  0000000C.00000005
CHF$L_SIG_ARG1          FFFFFFFF.8244BEF0  FFFFFFFC.00010000  SYS$K_VERSION_08
                        FFFFFFFF.8244BEF8  00000300.FFFFFFFC  UCB$L_PI_SVA
                        FFFFFFFF.8244BF00  00000002.00000001
                        FFFFFFFF.8244BF08  00000000.0000000C
                        FFFFFFFF.8244BF10  00000000.00000000
                        FFFFFFFF.8244BF18  00000000.FFFFFFFC
                        FFFFFFFF.8244BF20  00000008.00000000
                        FFFFFFFF.8244BF28  00000000.00000001
                        FFFFFFFF.8244BF30  00000008.00000000
                        FFFFFFFF.8244BF38  00000000.FFFFFFFC
INTSTK$Q_R2             FFFFFFFF.8244BF40  FFFFFFFF.80404668  SCH$GL_ACTIVE_PRIORITY
INTSTK$Q_R3             FFFFFFFF.8244BF48  FFFFFFFF.8042F280  SCH$WAIT_KERNEL_MODE
INTSTK$Q_R4             FFFFFFFF.8244BF50  FFFFFFFF.80615F00
INTSTK$Q_R5             FFFFFFFF.8244BF58  00000000.00000000
INTSTK$Q_R6             FFFFFFFF.8244BF60  FFFFFFFF.805AE000
INTSTK$Q_R7             FFFFFFFF.8244BF68  00000000.00000000
INTSTK$Q_PC             FFFFFFFF.8244BF70  00000000.FFFFFFFC
INTSTK$Q_PS             FFFFFFFF.8244BF78  30000000.00000300  UCB$L_PI_SVA
                        FFFFFFFF.8244BF80  FFFFFFFF.80404668  SCH$GL_ACTIVE_PRIORITY
                        FFFFFFFF.8244BF88  00000000.7FFD00A8  PIO$GW_IIOIMPA
                        FFFFFFFF.8244BF90  00000000.00000000
                        FFFFFFFF.8244BF98  FFFFFFFF.8042CD50  SCH$WAIT_PROC+00060
                        FFFFFFFF.8244BFA0  00000000.00000044
                        FFFFFFFF.8244BFA8  FFFFFFFF.80403C30  SMP$GL_FLAGS
Prev SP (8244BFB0) =>   FFFFFFFF.8244BFB0  FFFFFFFF.8042CD50  SCH$WAIT_PROC+00060
                        FFFFFFFF.8244BFB8  00000000.00000000
                        FFFFFFFF.8244BFC0  FFFFFFFF.805EE040
                        FFFFFFFF.8244BFC8  FFFFFFFF.8006DB54  PROCESS_MANAGEMENT_NPRO+0DB54
                        FFFFFFFF.8244BFD0  FFFFFFFF.80404668  SCH$GL_ACTIVE_PRIORITY
                        FFFFFFFF.8244BFD8  FFFFFFFF.80615F00
                        FFFFFFFF.8244BFE0  FFFFFFFF.8041B220  SCH$RESOURCE_WAIT
                        FFFFFFFF.8244BFE8  00000000.00000044
                        FFFFFFFF.8244BFF0  FFFFFFFF.80403C30  SMP$GL_FLAGS
                        FFFFFFFF.8244BFF8  00000000.7FF95E00
```

The SHOW STACK command displays a system stack. The data shown above the stack pointer may not be valid. Note that the mechanism array, signal array, and exception frame symbols displayed on the left will appear only for INVEXCEPTN, FATALEXCPT, UNXSIGNAL, and SSRVEXCEPT bugchecks.

SHOW SUMMARY

Displays a list of all active processes and the values of the parameters used in swapping and scheduling these processes.

Format

SHOW SUMMARY [/IMAGE|/THREAD]

Parameters

None.

Qualifiers

/IMAGE
Causes SDA to display, if possible, the name of the image being executed within each process.

/THREAD
Displays information on all the current threads associated with the current process.

Description

The SHOW SUMMARY command displays the information in Table SDA–32 for each active process in the system.

Table SDA–32 Process Information in the SHOW SUMMARY Display

Column	Contents
Extended PID	The 32-bit number that uniquely identifies the process
Indx	Index of this process into the PCB array
Process name	Name assigned to the process
Username	Name of the user who created the process
State	Current state of the process. Table SDA–33 shows the 14 states and their meanings.
Pri	Current scheduling priority of the process
PCB/KTB	Address of the process control block or address of the kernel thread block
PHD/FRED	Address of the process header or address of the floating-point registers and execution data block
Wkset	Number (in decimal) of pages currently in the process working set

Table SDA–33 Current State Information

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	Miscellaneous wait.
RWxxx	Waiting for system resource xxx.

Example

```
SDA> SHOW SUMMARY
Current process summary
-----------------------
 Extended Indx Process name    Username    State   Pri PCB/KTB  PHD/FRED Wkset
-- PID -- ---- --------------- ----------- ------- --- -------- -------- ------
 00000041 0001 SWAPPER                      HIB     16 80C641D0 80C63E00   0
 00000045 0005 IPCACP          SYSTEM       HIB     10 80DC0780 81266000  39
 00000046 0006 ERRFMT          SYSTEM       HIB      8 80DC2240 8126C000  57
 00000047 0007 OPCOM           SYSTEM       HIB      8 80DC3340 81272000  31
 00000048 0008 AUDIT_SERVER    AUDIT$SERVER HIB     10 80D61280 81278000 152
 00000049 0009 JOB_CONTROL     SYSTEM       HIB     10 80D620C0 8127E000  50
 0000004A 000A SECURITY_SERVER SYSTEM       HIB     10 80DC58C0 81284000 253
 0000004B 000B TP_SERVER       SYSTEM       HIB     10 80DC8900 8128A000  75
 0000004C 000C NETACP          DECNET       HIB     10 80DBFE00 8125A000  78
 0000004D 000D EVL             DECNET       HIB      6 80DCA080 81290000  76
 0000004E 000E REMACP          SYSTEM       HIB      8 80DE4E00 81296000  14
 00000050 0010 DECW$SERVER_0   SYSTEM       HIB      8 80DEF940 812A2000 739
 00000051 0011 DECW$LOGINOUT   <login>      LEF      4 80DF0F00 812A8000 273
 00000052 0012 SYSTEM          SYSTEM       LEF      9 80D772C0 81260000  75
```

The SHOW SUMMARY command describes all active processes in the system at the time of the system failure. Note that there was no process in the in the CUR state at the time of the failure.

SHOW SYMBOL

Displays the hexadecimal value of a symbol and, if the value is equal to an address location, the contents of that location.

Format

SHOW SYMBOL [/ALL] symbol-name

Parameter

symbol-name
Name of the symbol to be displayed. You must provide a **symbol-name**.

Qualifier

/ALL
Displays information on all symbols whose names begin with the characters specified in **symbol-name**.

Description

The SHOW SYMBOL/ALL command is useful for determining the values of symbols that belong to a symbol set, as illustrated in the following examples.

Examples

1.
```
SDA> SHOW SYMBOL G
G = FFFFFFFF.80000000 :  6BFA8001.201F0104
```

The SHOW SYMBOL command evaluates the symbol G as 80000000_{16} and displays the contents of address 80000000_{16} as $201F0104_{16}$.

2.
```
SDA>  SHOW SYMBOL/ALL BUG

Symbols sorted by name
----------------------
BUG$L_BUGCHK_FLAGS          = FFFFFFFF.804031E8 : 00000000.00000001
BUG$L_FATAL_SPSAV           = FFFFFFFF.804031F0 : 00000000.00000001
BUG$REBOOT                  = FFFFFFFF.8042E320 : 00000000.00001808
BUG$REBOOT_C                = FFFFFFFF.8004f4D0 : 47FB041D.47FD0600

   .
   .
   .
Symbols sorted by value
-----------------------
BUG$REBOOT_C                = FFFFFFFF.8004f4D0 :47FB041D.47FD0600
BUG$L_BUGCHK_FLAGS          = FFFFFFFF.804031E8 :00000000.00000001
BUG$L_FATAL_SPSAV           = FFFFFFFF.804031F0 :00000000.00000001
BUG$REBOOT                  = FFFFFFFF.8042E320 :00000000.00001808
   .
   .
   .
```

This example shows the display produced by the SHOW SYMBOL/ALL command. SDA searches its symbol table for all symbols that begin with the string "BUG" and displays the symbols and their values. Although certain values equate to memory addresses, it is doubtful that the contents of those addresses are actually relevant to the symbol definitions in this instance.

SHOW WORKING_SET_LIST

Displays the system working set list and retains the current process context.

Format

SHOW WORKING_SET_LIST [={GPT | SYSTEM | LOCKED | *n*}]

Parameters

None.

Qualifiers

None.

Description

The SHOW WORKING_SET_LIST command displays the contents of requested entries in the system working set list. If no option is given, all working set list entries are displayed. Table SDA–34 shows the options available with SHOW WORKING_SET_LIST. The SHOW WORKING_SET_LIST command is equivalent to the SHOW PROCESS/SYSTEM/WORKING_SET_LIST command. See the SHOW PROCESS command and Table SDA–25 for more information.

Table SDA–34 Options for the SHOW WORKING_SET_LIST Command

Options	Results
GPT	Displays only working set list entries that are for global page table pages.
SYSTEM	Displays only working set list entries for pageable system pages.
LOCKED	Displays only working set list entries for pageable system pages that are locked in the system working set.
n	Displays a specific working set entry, where *n* is the working set list index (WSLX) of the entry of interest.

SPAWN

Creates a subprocess of the process currently running SDA, copying the context of the current process to the subprocess and, optionally, executing a specified command within the subprocess.

Format

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

Parameter

command
Name of the command that you want the subprocess to execute.

Qualifiers

/INPUT=filespec
Specifies an input file containing one or more command strings to be executed by the spawned subprocess. If you specify a command string with an input file, the command string is processed before the commands in the input file. Once processing is complete, the subprocess is terminated.

/NOLOGICAL_NAMES
Specifies that the logical names of the parent process are not to be copied to the subprocess. The default behavior is that the logical names of the parent process are copied to the subprocess.

/NOSYMBOLS
Specifies that the DCL global and local symbols of the parent process are not to be passed to the subprocess. The default behavior is that these symbols are passed to the subprocess.

/NOTIFY
Specifies that a message is to be broadcast to SYS$OUTPUT when the subprocess completes processing or aborts. The default behavior is that such a message is not sent to SYS$OUTPUT.

/NOWAIT
Specifies that the system is not to wait until the subprocess is completed before allowing more commands to be specified. This qualifier allows you to specify new commands while the spawned subprocess is running. If you specify /NOWAIT, use /OUTPUT to direct the output of the subprocess to a file to prevent more than one process from simultaneously using your terminal.

The default behavior is that the system waits until the subprocess is completed before allowing more commands to be specified.

/OUTPUT=filespec
Specifies an output file to which the results of the SPAWN operation are written. To prevent output from the spawned subprocess from being displayed while you are specifying new commands, specify an output other than SYS$OUTPUT whenever you specify /NOWAIT. If you omit the /OUTPUT qualifier, output is written to the current SYS$OUTPUT device.

/PROCESS=*process-name*

Specifies the name of the subprocess to be created. The default name of the subprocess is *USERNAME_n*, where *USERNAME* is the user name of the parent process. The variable *n* represents the subprocess number.

Example

```
SDA>  SPAWN
$  MAIL
   .
   .
   .
$  DIR
   .
   .
   .
$  LO
   Process SYSTEM_1 logged out at 5-JAN-1993 15:42:23.59
SDA>
```

This example uses the SPAWN command to create a subprocess that issues DCL commands to invoke the Mail utility. The subprocess then lists the contents of a directory before logging out to return to the parent process executing SDA.

VALIDATE PFN_LIST

Validates that the page counts on lists are correct.

Format

VALIDATE PFN_LIST {/ALL (d)|[/BAD|/FREE|/MODIFIED|/ZERO]}

Parameters

None

Qualifiers

/ALL
Validates all the PFN lists: bad, free, modified, and zero.

/BAD
Validates the bad page list.

/FREE
Validates the free page list.

/MODIFIED
Validates the modified page list.

/ZERO
Validates the zero page list.

Description

The VALIDATES PFN_LIST command validates the specified PFN list(s) bit counting the number of entries in the list and comparing that to the running count of entries for each list maintained by the system.

Examples

1.
```
SDA> VALIDATES PFN_LIST/ALL
Free list: expected 445 pages, found 0 pages
         excluding zeroded free list with expected size 116 pages
Zeroed free list validated: 116 pages
Modified list validated: 311 pages
Bad page list validated: 0 pages
```

2.
```
SDA>VALIDATES PFN_LIST/FREE
Free list: expected 445 pages, found 0 pages
         excluding zeroed free list with expected size 116 pages
```

VALIDATE QUEUE

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

Format

VALIDATE QUEUE [address]
[/LIST | /QUADWORD | /SELF_RELATIVE | /SINGLY_LINKED]

Parameter

address
Address of an element in a queue.

If you specify the period (.) as the **address**, SDA uses the last evaluated expression as the queue element's address.

If you do not specify an **address**, the VALIDATE QUEUE command determines the address from the last issued VALIDATE QUEUE command in the current SDA session.

If you do not specify an **address**, and no queue has previously been specified, SDA displays the following error message:

```
%SDA-E-NOQUEUE, no queue has been specified for validation
```

Qualifiers

/LIST
Displays address of each element in the queue.

/QUADWORD
Allows the validate operation to occur on queues with linked lists of quadword addresses.

/SELF_RELATIVE
Specifies that the selected queue is a self-relative queue. Other processes cannot insert or remove queue entries while the current process is doing so.

/SINGLY_LINKED
Allows validation of queues that have no backward pointers.

Description

The VALIDATE QUEUE command uses the forward, and optionally, backward pointers in each element of the queue to make sure that all such pointers are valid and that the integrity of the queue is intact. If the queue is intact, SDA displays the following message:

```
Queue is complete, total of n elements in the queue
```

In these messages, *n* represents the number of entries the VALIDATE QUEUE command has found in the queue.

If SDA discovers an error in the queue, it displays one of the following error messages:

```
Error in forward queue linkage at address nnnnnnnn after tracing x elements
Error comparing backward link to previous structure address (nnnnnnnn)
Error occurred in queue element at address oooooooo after tracing pppp elements
```

These messages can appear frequently when the VALIDATE QUEUE command is used within an SDA session that is analyzing a running system. In a running system, the composition of a queue can change while the command is tracing its links, thus producing an error message.

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

```
The queue is empty
```

Examples

1.
```
SDA>  VALIDATE QUEUE/SELF_RELATIVE IOC$GQ_POSTIQ
Queue is complete, total of 159 elements in the queue
```

   This example validates the self-relative queue IOC$GQ_POSTIQ. The validation is successful and determines that there are 159 IRPs in the list.

2.
```
SDA> validate queue/quad FFFFFFFF80D0E6CO/list
Entry      Address              Flink                Blink
-----      -------              ------               -----
Header     FFFFFFFF80D0E6CO     FFFFFFFF80D03780     FFFFFFFF80D0E800
    1.     FFFFFFFF80D0E790     FFFFFFFF80D0E7CO     FFFFFFFF80D0E6C0
    2.     FFFFFFFF80D0E800     FFFFFFFF80D0E6C0     FFFFFFFF80D0E7C0
Queue is complete, total of 3 elements in the queue
```

   This example shows the validation of quadword elements in a list.

3.
```
SDA> validate queue/sing exe$gl_nonpaged+4
Queue is zero-terminated, total of 95 elements in the queue
```

   This example shows the validation of singly linked elements in the queue. The forward link of the final element is zero instead of being a pointer back to the queue header.

SDA Extension Commands

The SDA CLUE (Crash Log Utility Extractor) extension commands can summarize information provided by certain standard SDA commands and provide additional detail for some SDA commands. These SDA CLUE commands can interpret the contents of the dump to perform additonal analysis.

All CLUE commands can be used when analyzing crash dumps; the only CLUE commands that are not allowed when analyzing a running system are CLUE CRASH, CLUE ERRLOG, CLUE HISTORY, and CLUE STACK.

When rebooting after a system failure, CLUE commands by default automatically analyze and save summary information from the crash dump file in CLUE history and listing files. This information includes the following:

- Crash dump summary information
- System configuration
- Stack decoder
- Page and swap files
- Memory management statistics
- Process DCL recall buffer
- Active XQP processes
- XQP cache header

For additional information on the contents of the CLUE listing file, see the reference section on CLUE HISTORY.

The following SDA CLUE extension commands are described in this section:

CLUE CLEANUP
CLUE CONFIG
CLUE CRASH
CLUE ERRLOG
CLUE HISTORY
CLUE MCHK
CLUE MEMORY
CLUE PROCESS
CLUE STACK
CLUE VCC
CLUE XQP

CLUE CLEANUP

Performs housekeeping operations to conserve disk space.

Format

CLUE CLEANUP

Parameters

None.

Qualifiers

None.

Description

CLUE CLEANUP performs housekeeping operations to conserve disk space. To avoid filling up the system disk with listing files generated by CLUE, CLUE CLEANUP is run during system startup to check the overall disk space used by all CLUE$*.LIS files.

If the CLUE$COLLECT:CLUE$*.LIS files occupy more space than the logical CLUE$MAX_BLOCKS allows, then the oldest files are deleted until the threshold is reached. If this logical name is not defined, a default value of 5,000 disk blocks is assumed. A value of zero disables housekeeping and no check on the disk space is performed.

Example

```
SDA> CLUE CLEANUP
%CLUE-I-CLEANUP, housekeeping started...
%CLUE-I-MAXBLOCK, maximum blocks allowed 5000 blocks
%CLUE-I-STAT, total of 4 CLUE files, 192 blocks.
%CLUE-I-DEL, deleting DISK$X6AF_G5N:[SYSCOMMON.SYSERR]CLUE$_010193_0000.LIS;1 (78 blocks)
```

In this example, the CLUE CLEANUP command displays that the total number of blocks of disk space used by CLUE files does not exceed the maximum number of blocks allowed. No files are deleted.

CLUE CONFIG

Displays the system, memory, and device configurations.

Format

CLUE CONFIG

Parameters

None.

Qualifiers

None.

Description

CLUE CONFIG displays the system, memory, and device configurations.

Example

```
SDA> CLUE CONFIG
System Configuration:
---------------------
System Information:
System Type    ALPHAbook 1                          Primary CPU ID 00
Cycle Time     8.6 nsec (115 MHz)                   Pagesize       8192 Byte

Memory Configuration:
Cluster    PFN Start    PFN Count         Range (MByte)        Usage
 #03             0          256         0.0 MB -     2.0 MB    Console
 #04           256         7935         2.0 MB -    63.9 MB    System
 #05          8191            1        63.9 MB -    64.0 MB    Console

Per-CPU Slot Processor Information:
CPU ID         00                       CPU State    rc,pa,pp,cv,pv,pmv,pl
CPU Type       LCA  Pass 2 (21066)      Halt PC      00000000.20000000
PAL Code       5.56                     Halt PS      00000000.00001F00
CPU Revision   ....                     Halt Code    00000000.00000000
Serial Number  ..........               "Bootstrap or Powerfail"
Console Vers   V4.6-29
```

```
Adapter Configuration:
----------------------
TR Adapter     ADP      Hose Bus   BusArrayEntry  Node Device Name / HW-Id
-- ----------- -------- ---- ------------------- ---- -------------------------
 1 KA1504      80D6F680    0 BUSLESS_SYSTEM
 2 PCI         80D6F880    0 PCI
                             80D6FBE8  PKA:    6 NCR 53C810 SCSI
                             80D6FC20          7 SATURN
                             80D6FC58          8 PCMCIA_PD6729
 3 ISA         80D6FE80    0 ISA
                             80D70098          0 EISA_SYSTEM_BOARD
                             80D700D0  AUA:    1 PCXBJ
                             80D70108  GQA:    2 AlphaBOOK-1 LCD (WD90C24A)
                             80D70140  HEA:    3 H8 AlphaBook-I uProc
 4 XBUS        80D70440    0 XBUS
                             80D70618          0 MOUS
                             80D70650          1 KBD
                             80D70688  TTA:    2 NS16450 Serial Port
                             80D706C0  LRA:    3 Line Printer (parallel port)
                             80D706F8  DVA:    4 Floppy
 5 PCMCIA      80D71040    0 PCMCIA
                             80D71218  EOA:    0 3Com Etherlink III
```

CLUE CRASH

Displays a crash dump summary.

Format

CLUE CRASH

Parameters

None.

Qualifiers

None.

Description

CLUE CRASH displays a crash dump summary, which includes the following items:

- Bugcheck type
- Current process and image
- Failing PC and PS
- Executive image section name and offset
- General registers
- Failing instructions
- Exception frame, signal and mechanism arrays (if available)

Example

```
SDA> CLUE CRASH
Crash Time:        30-AUG-1996 13:13:46.83
Bugcheck Type:     SSRVEXCEPT, Unexpected system service exception
Node:              SWPCTX  (Standalone)
CPU Type:          DEC 3000 Model 400
VMS Version:       X6AF-FT2
Current Process:   SYSTEM
Current Image:     $31$DKB0:[SYS0.][SYSMGR]X.EXE;1
Failing PC:        00000000.00030078    SYS$K_VERSION_01+00078
Failing PS:        00000000.00000003
Module:            X
Offset:            00030078

Boot Time:         30-AUG-1996 09:06:22.00
System Uptime:              0 04:07:24.83
Crash/Primary CPU: 00/00
System/CPU Type:   0402
Saved Processes:   18
Pagesize:          8 KByte (8192 bytes)
Physical Memory:   64 MByte (8192 PFNs, contiguous memory)
Dumpfile Pagelets: 98861 blocks
Dump Flags:        olddump,writecomp,errlogcomp,dump_style
Dump Type:         raw,selective
EXE$GL_FLAGS:      poolpging,init,bugdump
Paging Files:      1 Pagefile and 1 Swapfile installed
```

```
Stack Pointers:
KSP = 00000000.7FFA1C98   ESP = 00000000.7FFA6000   SSP = 00000000.7FFAC100
USP = 00000000.7AFFBAD0

General Registers:
R0  = 00000000.00000000   R1  = 00000000.7FFA1EB8   R2  = FFFFFFFF.80D0E6C0
R3  = FFFFFFFF.80C63460   R4  = FFFFFFFF.80D12740   R5  = 00000000.000000C8
R6  = 00000000.00030038   R7  = 00000000.7FFA1FC0   R8  = 00000000.7FFAC208
R9  = 00000000.7FFAC410   R10 = 00000000.7FFAD238   R11 = 00000000.7FFCE3E0
R12 = 00000000.00000000   R13 = FFFFFFFF.80C6EB60   R14 = 00000000.00000000
R15 = 00000000.009A79FD   R16 = 00000000.000003C4   R17 = 00000000.7FFA1D40
R18 = FFFFFFFF.80C05C38   R19 = 00000000.00000000   R20 = 00000000.7FFA1F50
R21 = 00000000.00000000   R22 = 00000000.00000001   R23 = 00000000.7FFF03C8
R24 = 00000000.7FFF0040   AI  = 00000000.00000003   RA  = FFFFFFFF.82A21080
PV  = FFFFFFFF.829CF010   R28 = FFFFFFFF.8004B6DC   FP  = 00000000.7FFA1CA0
PC  = FFFFFFFF.82A210B4   PS  = 18000000.00000000

Exception Frame:
R2  = 00000000.00000003   R3  = FFFFFFFF.80C63460   R4  = FFFFFFFF.80D12740
R5  = 00000000.000000C8   R6  = 00000000.00030038   R7  = 00000000.7FFA1FC0
PC  = 00000000.00030078   PS  = 00000000.00000003

Signal Array:                          64-bit Signal Array:
Arg Count    = 00000005                Arg Count      =          00000005
Condition    = 0000000C                Condition      = 00000000.0000000C
Argument #2  = 00010000                Argument #2    = 00000000.00010000
Argument #3  = 00000000                Argument #3    = 00000000.00000000
Argument #4  = 00030078                Argument #4    = 00000000.00030078
Argument #5  = 00000003                Argument #5    = 00000000.00000003

Mechanism Array:
Arguments    = 0000002C                Establisher FP = 00000000.7AFFBAD0
Flags        = 00000000                Exception FP   = 00000000.7FFA1F00
Depth        = FFFFFFFD                Signal Array   = 00000000.7FFA1EB8
Handler Data = 00000000.00000000       Signal64 Array = 00000000.7FFA1ED0
R0  = 00000000.00020000   R1  = 00000000.00000000   R16 = 00000000.00020004
R17 = 00000000.00010050   R18 = FFFFFFFF.FFFFFFFF   R19 = 00000000.00000000
R20 = 00000000.7FFA1F50   R21 = 00000000.00000000   R22 = 00000000.00010050
R23 = 00000000.00000000   R24 = 00000000.00010051   R25 = 00000000.00000000
R26 = FFFFFFFF.8010ACA4   R27 = 00000000.00010050   R28 = 00000000.00000000

System Registers:
Page Table Base Register (PTBR)                          00000000.00001136
Processor Base Register (PRBR)                           FFFFFFFF.80D0E000
Privileged Context Block Base (PCBB)                     00000000.003FE080
System Control Block Base (SCBB)                         00000000.000001DC
Software Interrupt Summary Register (SISR)               00000000.00000000
Address Space Number (ASN)                               00000000.0000002F
AST Summary / AST Enable (ASTSR_ASTEN)                   00000000.0000000F
Floating-Point Enable (FEN)                              00000000.00000000
Interrupt Priority Level (IPL)                           00000000.00000000
Machine Check Error Summary (MCES)                       00000000.00000000
Virtual Page Table Base Register (VPTB)                  FFFFFFFC.00000000

Failing Instruction:
SYS$K_VERSION_01+00078:         LDL             R28,(R28)
```

```
Instruction Stream (last 20 instructions):
SYS$K_VERSION_01+00028:         LDQ             R16,#X0030(R13)
SYS$K_VERSION_01+0002C:         LDQ             R27,#X0048(R13)
SYS$K_VERSION_01+00030:         LDA             R17,(R28)
SYS$K_VERSION_01+00034:         JSR             R26,(R26)
SYS$K_VERSION_01+00038:         LDQ             R26,#X0038(R13)
SYS$K_VERSION_01+0003C:         BIS             R31,SP,SP
SYS$K_VERSION_01+00040:         BIS             R31,R26,R0
SYS$K_VERSION_01+00044:         BIS             R31,FP,SP
SYS$K_VERSION_01+00048:         LDQ             R28,#X0008(SP)
SYS$K_VERSION_01+0004C:         LDQ             R13,#X0010(SP)
SYS$K_VERSION_01+00050:         LDQ             FP,#X0018(SP)
SYS$K_VERSION_01+00054:         LDA             SP,#X0020(SP)
SYS$K_VERSION_01+00058:         RET             R31,(R28)
SYS$K_VERSION_01+0005C:         BIS             R31,R31,R31
SYS$K_VERSION_01+00060:         LDA             SP,#XFFE0(SP)
SYS$K_VERSION_01+00064:         STQ             FP,#X0018(SP)
SYS$K_VERSION_01+00068:         STQ             R27,(SP)
SYS$K_VERSION_01+0006C:         BIS             R31,SP,FP
SYS$K_VERSION_01+00070:         STQ             R26,#X0010(SP)
SYS$K_VERSION_01+00074:         LDA             R28,(R31)
SYS$K_VERSION_01+00078:         LDL             R28,(R28)
SYS$K_VERSION_01+0007C:         BEQ             R28,#X000007
SYS$K_VERSION_01+00080:         LDQ             R26,#XFFE8(R27)
SYS$K_VERSION_01+00084:         BIS             R31,R26,R0
SYS$K_VERSION_01+00088:         BIS             R31,FP,SP
```

CLUE ERRLOG

Extracts the error log buffers from the dump file and places them into the binary file called CLUE$ERRLOG.SYS.

Format

CLUE ERRLOG

Parameters

None.

Qualifiers

None.

Description

CLUE ERRLOG extracts the error log buffers from the dump file and places them into the binary file called CLUE$ERRLOG.SYS.

These buffers contain messages not yet written to the error log file at the time of the failure. When you analyze a failure on the same system on which it occurred, you can run the Error Log utility on the actual error log file to see these error log messages. When analyzing a failure from another system, use the CLUE ERRLOG command to create a file containing the failing system's error log messages just prior to the failure. System failures are often triggered by hardware problems, so determining what, if any, hardware errors occurred prior to the failure can help you troubleshoot a failure.

You can define the logical CLUE$ERRLOG to any file specification if you want error log information written to a file other than CLUE$ERRLOG.SYS.

Example

```
SDA> CLUE ERRLOG

Sequence  Date        Time
--------  ----------- -----------
     128  11-MAY-1994 00:39:31.30
     129  11-MAY-1994 00:39:32.12
     130  11-MAY-1994 00:39:44.83
     131  11-MAY-1994 00:44:38.97 * Crash Entry
```

The CLUE ERRLOG command diplays the sequence, date, and time of each error log buffer extracted from a dump file in the file CLUE$ERRLOG.SYS.

CLUE HISTORY

Updates history file and generates crash dump summary output.

Format

CLUE HISTORY [/qualifier]

Parameters

None.

Qualifier

/OVERRIDE
Allows execution of this command even if the dump file has already been analyzed (DMP$V_OLDDUMP bit set).

Description

This command updates the history file pointed to by the logical name CLUE$HISTORY with a one-line entry and the major crash dump summary information. If CLUE$HISTORY is not defined, a file CLUE$HISTORY.DAT in your default directory will be created.

In addition, a listing file with summary information about the system failure is created in the directory pointed to by CLUE$COLLECT. The file name is of the form CLUE$*node_ddmmyy_hhmm*.LIS where the timestamp (*hhmm*) corresponds to the system failure time and not the time when the file was created.

The listing file contains summary information collected from the following SDA commands:

- CLUE CRASH
- CLUE CONFIG
- CLUE MEMORY/FILES
- CLUE MEMORY/STATISTIC
- CLUE PROCESS/RECALL
- CLUE XQP/ACTIVE

Refer to the reference section for each of these commands to see examples of the displayed information.

The logical name CLUE$FLAG controls how much information is written to the listing file.

- Bit 0—Include crash dump summary
- Bit 1—Include system configuration
- Bit 2—Include stack decoding information
- Bit 3—Include page and swap file usage
- Bit 4—Include memory management statistics
- Bit 5—Include process DCL recall buffer

- Bit 6—Include active XQP process information
- Bit 7—Include XQP cache header

If this logical name is undefined, all bits are set by default internally and all information is written to the listing file. If the value is zero, no listing file is generated. The value has to be supplied in hexadecimal form (for example, DEFINE CLUE$FLAG 81 will include the crash dump summary and the XQP cache header information).

If the logical name CLUE$SITE_PROC points to a valid and existing file, it will be executed as part of the CLUE HISTORY command (for example, automatic saving of the dump file during system startup). If used, this file should contain only valid SDA commands.

Refer to Section 1.3 for more information on site-specific command files.

CLUE MCHK

This command is obsolete.

Format

CLUE MCHK

Parameters

None.

Qualifiers

None.

Description

The CLUE MCHK command has been withdrawn. Issuing the command produces the following output, explaining the correct way to obtain MACHINECHECK information from a crash dump.

Please use the following commands in order to extract the errorlog buffers from the dumpfile header and analyze the machine check entry:

```
$ analyze/crash sys$system:sysdump.dmp
SDA> clue errlog
SDA> exit
$ diagnose clue$errlog
```

CLUE MEMORY

Displays memory- and pool-related information.

Format

CLUE MEMORY [/qualifier[,...]]

Parameters

None.

Qualifiers

/FILES
Displays information about page and swap file usage.

/FREE [/FULL]
Validates and displays dynamic nonpaged free packet list queue.

/GH [/FULL]
Displays information about the granularity hint regions.

/LAYOUT
Decodes and displays much of the system virtual address space layout.

/LOOKASIDE
Validates the lookaside list queue heads and counts the elements for each list.

/STATISTIC
Displays systemwide performance data such as page fault, I/O, pool, lock manager, MSCP, and file system cache.

Description

The CLUE MEMORY command displays memory- and pool-related information.

Examples

1.
```
SDA> CLUE MEMORY/FILES
Paging File Usage (blocks):
---------------------------
Swapfile (Index  1)                     Device                    DKB0:
   PFL Address      FFFFFFFF.80D74A80    UCB Address      FFFFFFFF.80D53940
   Free Blocks                 4992     Reserved Blocks             4992
   Total Size (blocks)        10112     Flags                      inited
   Paging Usage (processes)       0     Swap Usage (processes)         5
   Alloc Size SWPINC (pages)     64     Largest Chunk (pages)        184
   Chunks GEQ SWPINC              3     Chunks LT SWPINC               0

Pagefile (Index  3)                     Device                    DKB0:
   PFL Address      FFFFFFFF.80D74600    UCB Address      FFFFFFFF.80D53940
   Free Blocks               108208     Reserved Blocks            37808
   Total Size (blocks)       139008     Flags                      inited
   Paging Usage (processes)      21     Swap Usage (processes)         0
   Alloc Size SWPINC (pages)     64     Largest Chunk (pages)       6576
   Chunks GEQ SWPINC              2     Chunks LT SWPINC               1

   Summary:  1 Pagefile and 1 Swapfile installed
```

This example shows the display produced by the CLUE MEMORY/FILES command.

2.
```
SDA> CLUE MEMORY/FREE/FULL
Non-Paged Dynamic Storage Pool - Variable Free Packet Queue:
------------------------------------------------------------

CLASSDR  FFFFFFFF.80D157C0 :    64646464 64646464 00000040 80D164C0  ÀdÑ.@...dddddddd
CLASSDR  FFFFFFFF.80D164C0 :    64646464 64646464 00000080 80D17200  .rÑ.....dddddddd
CLASSDR  FFFFFFFF.80D17200 :    64646464 64646464 00000080 80D21AC0  À.Ò.....dddddddd
CLASSDR  FFFFFFFF.80D21AC0 :    64646464 64646464 00000080 80D228C0  À(Ò.....dddddddd
VCC      FFFFFFFF.80D228C0 :    801CA5E8 026F0040 00000040 80D23E40  @>Ò.@...@.o.è¥..
CLASSDR  FFFFFFFF.80D23E40 :    64646464 64646464 00000040 80D24040  @@Ò.@...dddddddd
CLASSDR  FFFFFFFF.80D24040 :    64646464 64646464 00000040 80D26FC0  ÀoÒ.@...dddddddd
CLASSDR  FFFFFFFF.80D26FC0 :    64646464 64646464 00000080 80D274C0  ÀtÒ.....dddddddd
CLASSDR  FFFFFFFF.80D274C0 :    64646464 64646464 00000040 80D2E200  .âÒ.@...dddddddd
CLASSDR  FFFFFFFF.80D2E200 :    64646464 64646464 00000080 80D2E440  @äÒ.....dddddddd
CLASSDR  FFFFFFFF.80D2E440 :    64646464 64646464 00000040 80D2F000  .Ò.@...dddddddd
CLASSDR  FFFFFFFF.80D2F000 :    64646464 64646464 00000080 80D2F400  .ôÒ.....dddddddd
   .
   .
   .
CLASSDR  FFFFFFFF.80E91D40 :    64646464 64646464 00000500 80E983C0  À.é.....dddddddd
CLASSDR  FFFFFFFF.80E983C0 :    64646464 64646464 00031C40 00000000  ....@...dddddddd

Free Packet Queue, Status: Valid, 174 elements

   Largest free chunk:        00031C40 (hex)    203840 (dec)  bytes
   Total free dynamic space:  0003D740 (hex)    251712 (dec)  bytes
```

The CLUE MEMORY/FREE/FULL command validates and displays dynamic nonpaged free packet list queue.

3.
```
SDA> CLUE MEMORY/GH/FULL
Granularity Hint Regions - Huge Pages:
--------------------------------------

Execlet Code Region                                                Pages/Slices
   Base/End VA     FFFFFFFF.80000000 FFFFFFFF.80356000  Current Size  427/   427
   Base/End PA     00000000.00400000 00000000.00756000  Free            /     0
   Total Size      00000000.00356000            3.3 MB  In Use          /   427
   Bitmap VA/Size  FFFFFFFF.80D17CC0 00000000.00000040  Initial Size  512/   512
   Slice Size      00000000.00002000                    Released       85/    85
   Next free Slice 00000000.000001AB
```

```
   Image                          Base              End          Length
   SYS$PUBLIC_VECTORS          FFFFFFFF.80000000  FFFFFFFF.80001A00  00001A00
   SYS$BASE_IMAGE              FFFFFFFF.80002000  FFFFFFFF.8000D400  0000B400
   SYS$CNBTDRIVER              FFFFFFFF.8000E000  FFFFFFFF.8000F000  00001000
   SYS$NISCA_BTDRIVER          FFFFFFFF.80010000  FFFFFFFF.8001FA00  0000FA00
   SYS$ESBTDRIVER              FFFFFFFF.80020000  FFFFFFFF.80022400  00002400
   SYS$OPDRIVER                FFFFFFFF.80024000  FFFFFFFF.80027C00  00003C00
   SYSTEM_DEBUG                FFFFFFFF.80028000  FFFFFFFF.80050200  00028200
   SYSTEM_PRIMITIVES           FFFFFFFF.80052000  FFFFFFFF.80089000  00037000
   SYSTEM_SYNCHRONIZATION      FFFFFFFF.8008A000  FFFFFFFF.80095400  0000B400
   ERRORLOG                    FFFFFFFF.80096000  FFFFFFFF.80099200  00003200
   SYS$CPU_ROUTINES_0402       FFFFFFFF.8009A000  FFFFFFFF.800A3A00  00009A00
   EXCEPTION_MON               FFFFFFFF.800A4000  FFFFFFFF.800BC800  00018800
   IO_ROUTINES_MON             FFFFFFFF.800BE000  FFFFFFFF.800E2000  00024000
   SYSDEVICE                   FFFFFFFF.800E2000  FFFFFFFF.800E5C00  00003C00
   PROCESS_MANAGEMENT_MON      FFFFFFFF.800E6000  FFFFFFFF.8010B000  00025000
   SYS$VM                      FFFFFFFF.8010C000  FFFFFFFF.80167200  0005B200
   SHELL8K                     FFFFFFFF.80168000  FFFFFFFF.80169200  00001200
   LOCKING                     FFFFFFFF.8016A000  FFFFFFFF.8017BE00  00011E00
   MESSAGE_ROUTINES            FFFFFFFF.8017C000  FFFFFFFF.80182A00  00006A00
   LOGICAL_NAMES               FFFFFFFF.80184000  FFFFFFFF.80186C00  00002C00
   F11BXQP                     FFFFFFFF.80188000  FFFFFFFF.80190400  00008400
   SYSLICENSE                  FFFFFFFF.80192000  FFFFFFFF.80192400  00000400
   IMAGE_MANAGEMENT            FFFFFFFF.80194000  FFFFFFFF.80197A00  00003A00
   SECURITY                    FFFFFFFF.80198000  FFFFFFFF.801A0E00  00008E00
   SYSGETSYI                   FFFFFFFF.801A2000  FFFFFFFF.801A3A00  00001A00
   SYS$TRANSACTION_SERVICES    FFFFFFFF.801A4000  FFFFFFFF.801C5000  00021000
   SYS$UTC_SERVICES            FFFFFFFF.801C6000  FFFFFFFF.801C7000  00001000
   SYS$VCC_MON                 FFFFFFFF.801C8000  FFFFFFFF.801D4E00  0000CE00
   SYS$IPC_SERVICES            FFFFFFFF.801D6000  FFFFFFFF.80214A00  0003EA00
   SYSLDR_DYN                  FFFFFFFF.80216000  FFFFFFFF.80219200  00003200
   SYS$MME_SERVICES            FFFFFFFF.8021A000  FFFFFFFF.8021B000  00001000
   SYS$TTDRIVER                FFFFFFFF.8021C000  FFFFFFFF.8022FE00  00013E00
   SYS$PKCDRIVER               FFFFFFFF.80230000  FFFFFFFF.80240400  00010400
   SYS$DKDRIVER                FFFFFFFF.80242000  FFFFFFFF.80251600  0000F600
   RMS                         FFFFFFFF.80252000  FFFFFFFF.802C5E00  00073E00
   SYS$GXADRIVER               FFFFFFFF.802C6000  FFFFFFFF.802CE000  00008000
   SYS$ECDRIVER                FFFFFFFF.802CE000  FFFFFFFF.802D1000  00003000
   SYS$LAN                     FFFFFFFF.802D2000  FFFFFFFF.802D8E00  00006E00
   SYS$LAN_CSMACD              FFFFFFFF.802DA000  FFFFFFFF.802E6600  0000C600
   SYS$MKDRIVER                FFFFFFFF.802E8000  FFFFFFFF.802F1C00  00009C00
   SYS$YRDRIVER                FFFFFFFF.802F2000  FFFFFFFF.802F9600  00007600
   SYS$SODRIVER                FFFFFFFF.802FA000  FFFFFFFF.802FF000  00005000
   SYS$INDRIVER                FFFFFFFF.80300000  FFFFFFFF.8030EA00  0000EA00
   NETDRIVER                   FFFFFFFF.80310000  FFFFFFFF.80310200  00000200
   NETDRIVER                   FFFFFFFF.80312000  FFFFFFFF.80329E00  00017E00
   SYS$IMDRIVER                FFFFFFFF.8032A000  FFFFFFFF.8032EA00  00004A00
   SYS$IKDRIVER                FFFFFFFF.80330000  FFFFFFFF.8033AC00  0000AC00
   NDDRIVER                    FFFFFFFF.8033C000  FFFFFFFF.8033F800  00003800
   SYS$WSDRIVER                FFFFFFFF.80340000  FFFFFFFF.80341600  00001600
   SYS$CTDRIVER                FFFFFFFF.80342000  FFFFFFFF.8034D200  0000B200
   SYS$RTTDRIVER               FFFFFFFF.8034E000  FFFFFFFF.80351800  00003800
   SYS$FTDRIVER                FFFFFFFF.80352000  FFFFFFFF.80354200  00002200

Execlet Data Region                                                 Pages/Slices
   Base/End VA     FFFFFFFF.80C00000 FFFFFFFF.80CC0000  Current Size   96/  1536
   Base/End PA     00000000.00800000 00000000.008C0000  Free            /    11
   Total Size      00000000.000C0000            0.7 MB  In Use          /  1525
   Bitmap VA/Size  FFFFFFFF.80D17D00 00000000.00000100  Initial Size  128/  2048
   Slice Size      00000000.00000200                    Released       32/   512
   Next free Slice 00000000.000005F5
```

```
    Image                              Base              End          Length
    SYS$PUBLIC_VECTORS             FFFFFFFF.80C00000  FFFFFFFF.80C05000  00005000
    SYS$BASE_IMAGE                 FFFFFFFF.80C05000  FFFFFFFF.80C25E00  00020E00
    SYS$CNBTDRIVER                 FFFFFFFF.80C25E00  FFFFFFFF.80C26200  00000400
    SYS$NISCA_BTDRIVER             FFFFFFFF.80C26200  FFFFFFFF.80C29400  00003200
    SYS$ESBTDRIVER                 FFFFFFFF.80C29400  FFFFFFFF.80C29800  00000400
    SYS$OPDRIVER                   FFFFFFFF.80C29800  FFFFFFFF.80C2A200  00000A00
    SYSTEM_DEBUG                   FFFFFFFF.80C2A200  FFFFFFFF.80C4E400  00024200
    SYSTEM_PRIMITIVES              FFFFFFFF.80C4E400  FFFFFFFF.80C58200  00009E00
    SYSTEM_SYNCHRONIZATION         FFFFFFFF.80C58200  FFFFFFFF.80C5A000  00001E00
    ERRORLOG                       FFFFFFFF.80C5A000  FFFFFFFF.80C5A600  00000600
    SYS$CPU_ROUTINES_0402          FFFFFFFF.80C5A600  FFFFFFFF.80C5CA00  00002400
    EXCEPTION_MON                  FFFFFFFF.80C5CA00  FFFFFFFF.80C64C00  00008200
    IO_ROUTINES_MON                FFFFFFFF.80C64C00  FFFFFFFF.80C6AA00  00005E00
    SYSDEVICE                      FFFFFFFF.80C6AA00  FFFFFFFF.80C6B600  00000C00
    PROCESS_MANAGEMENT_MON         FFFFFFFF.80C6B600  FFFFFFFF.80C72600  00007000
    SYS$VM                         FFFFFFFF.80C72600  FFFFFFFF.80C79000  00006A00
    SHELL8K                        FFFFFFFF.80C79000  FFFFFFFF.80C7A000  00001000
    LOCKING                        FFFFFFFF.80C7A000  FFFFFFFF.80C7BA00  00001A00
    MESSAGE_ROUTINES               FFFFFFFF.80C7BA00  FFFFFFFF.80C7D000  00001600
    LOGICAL_NAMES                  FFFFFFFF.80C7D000  FFFFFFFF.80C7E200  00001200
    F11BXQP                        FFFFFFFF.80C7E200  FFFFFFFF.80C7FA00  00001800
    SYSLICENSE                     FFFFFFFF.80C7FA00  FFFFFFFF.80C7FE00  00000400
    IMAGE_MANAGEMENT               FFFFFFFF.80C7FE00  FFFFFFFF.80C80600  00000800
    SECURITY                       FFFFFFFF.80C80600  FFFFFFFF.80C83000  00002A00
    SYSGETSYI                      FFFFFFFF.80C83000  FFFFFFFF.80C83200  00000200
    SYS$TRANSACTION_SERVICES       FFFFFFFF.80C83200  FFFFFFFF.80C89E00  00006C00
    SYS$UTC_SERVICES               FFFFFFFF.80C89E00  FFFFFFFF.80C8A200  00000400
    SYS$VCC_MON                    FFFFFFFF.80C8A200  FFFFFFFF.80C8BC00  00001A00
    SYS$IPC_SERVICES               FFFFFFFF.80C8BC00  FFFFFFFF.80C91000  00005400
    SYSLDR_DYN                     FFFFFFFF.80C91000  FFFFFFFF.80C92200  00001200
    SYS$MME_SERVICES               FFFFFFFF.80C92200  FFFFFFFF.80C92600  00000400
    SYS$TTDRIVER                   FFFFFFFF.80C92600  FFFFFFFF.80C94C00  00002600
    SYS$PKCDRIVER                  FFFFFFFF.80C94C00  FFFFFFFF.80C96A00  00001E00
    SYS$DKDRIVER                   FFFFFFFF.80C96A00  FFFFFFFF.80C99800  00002E00
    RMS                            FFFFFFFF.80C99800  FFFFFFFF.80CAAC00  00011400
    RECOVERY_UNIT_SERVICES         FFFFFFFF.80CAAC00  FFFFFFFF.80CAB000  00000400
    SYS$GXADRIVER                  FFFFFFFF.80CAB000  FFFFFFFF.80CAF000  00004000
    SYS$ECDRIVER                   FFFFFFFF.80CAF000  FFFFFFFF.80CAFC00  00000C00
    SYS$LAN                        FFFFFFFF.80CAFC00  FFFFFFFF.80CB0800  00000C00
    SYS$LAN_CSMACD                 FFFFFFFF.80CB0800  FFFFFFFF.80CB1800  00001000
    SYS$MKDRIVER                   FFFFFFFF.80CB1800  FFFFFFFF.80CB3000  00001800
    SYS$YRDRIVER                   FFFFFFFF.80CB3000  FFFFFFFF.80CB3C00  00000C00
    SYS$SODRIVER                   FFFFFFFF.80CB3C00  FFFFFFFF.80CB4E00  00001200
    SYS$INDRIVER                   FFFFFFFF.80CB4E00  FFFFFFFF.80CB5E00  00001000
    NETDRIVER                      FFFFFFFF.80CB5E00  FFFFFFFF.80CB8800  00002A00
    SYS$IMDRIVER                   FFFFFFFF.80CB8800  FFFFFFFF.80CB9400  00000C00
    SYS$IKDRIVER                   FFFFFFFF.80CB9400  FFFFFFFF.80CBAA00  00001600
    NDDRIVER                       FFFFFFFF.80CBAA00  FFFFFFFF.80CBB400  00000A00
    SYS$WSDRIVER                   FFFFFFFF.80CBB400  FFFFFFFF.80CBBC00  00000800
    SYS$CTDRIVER                   FFFFFFFF.80CBBC00  FFFFFFFF.80CBD800  00001C00
    SYS$RTTDRIVER                  FFFFFFFF.80CBD800  FFFFFFFF.80CBE200  00000A00
    SYS$FTDRIVER                   FFFFFFFF.80CBE200  FFFFFFFF.80CBEA00  00000800
    11 free Slices                 FFFFFFFF.80CBEA00  FFFFFFFF.80CC0000  00001600

S0/S1 Executive Data Region                                        Pages/Slices
   Base/End VA     FFFFFFFF.80D00000 FFFFFFFF.80ECA000  Current Size  229/   229
   Base/End PA     00000000.00900000 00000000.00ACA000  Free             /     0
   Total Size      00000000.001CA000            1.7 MB  In Use           /   229
   Bitmap VA/Size  FFFFFFFF.80D17E00 00000000.00000020  Initial Size  229/   229
   Slice Size      00000000.00002000                    Released        0/     0
   Next free Slice 00000000.00000007
```

```
   Item                                Base              End          Length
   System Header                 FFFFFFFF.80D00000  FFFFFFFF.80D0A000  0000A000
   Error Log Allocation Buffers  FFFFFFFF.80D0A000  FFFFFFFF.80D0C000  00002000
   Nonpaged Pool (initial size)  FFFFFFFF.80D0E000  FFFFFFFF.80ECA000  001BC000

Resident Image Code Region                                          Pages/Slices
   Base/End VA     FFFFFFFF.80400000 FFFFFFFF.80C00000  Current Size 1024/  1024
   Base/End PA     00000000.00C00000 00000000.01400000  Free            /   223
   Total Size      00000000.00800000           8.0 MB   In Use          /   801
   Bitmap VA/Size  FFFFFFFF.80D17E20 00000000.00000080  Initial Size 1024/  1024
   Slice Size      00000000.00002000                    Released       0/     0
   Next free Slice 00000000.00000321

   Image                               Base              End          Length
   LIBRTL                        FFFFFFFF.80400000  FFFFFFFF.8049EA00  0009EA00
   LIBOTS                        FFFFFFFF.804A0000  FFFFFFFF.804AEC00  0000EC00
   CMA$TIS_SHR                   FFFFFFFF.804B0000  FFFFFFFF.804B2600  00002600
   DPML$SHR                      FFFFFFFF.804B4000  FFFFFFFF.8050B600  00057600
   DECC$SHR                      FFFFFFFF.8050C000  FFFFFFFF.80657000  0014B000
   SECURESHRP                    FFFFFFFF.80658000  FFFFFFFF.80676000  0001E000
   SECURESHR                     FFFFFFFF.80676000  FFFFFFFF.8068C000  00016000
   SECURESHR                     FFFFFFFF.8068C000  FFFFFFFF.8068C200  00000200
   LBRSHR                        FFFFFFFF.8068E000  FFFFFFFF.806A3E00  00015E00
   DECW$TRANSPORT_COMMON         FFFFFFFF.806A4000  FFFFFFFF.806B0C00  0000CC00
   CDE$UNIX_ROUTINES             FFFFFFFF.806B2000  FFFFFFFF.806C1E00  0000FE00
   DECW$XLIBSHR                  FFFFFFFF.806C2000  FFFFFFFF.80781C00  000BFC00
   DECW$XTLIBSHRR5               FFFFFFFF.80782000  FFFFFFFF.807C7600  00045600
   DECW$XMLIBSHR12               FFFFFFFF.807C8000  FFFFFFFF.8096AE00  001A2E00
   DECW$MRMLIBSHR12              FFFFFFFF.8096C000  FFFFFFFF.80994200  00028200
   DECW$DXMLIBSHR12              FFFFFFFF.80996000  FFFFFFFF.80A40400  000AA400
   223 free Slices               FFFFFFFF.80A42000  FFFFFFFF.80C00000  001BE000

S2 Executive Data Region                                            Pages/Slices
   Base/End VA     FFFFFFFE.00000000 FFFFFFFE.00050000  Current Size   40/     8
   Base/End PA     00000000.00350000 00000000.003A0000  Free            /     0
   Total Size      00000000.00050000           0.3 MB   In Use          /     8
   Bitmap VA/Size  FFFFFFFF.80D17EA0 00000000.00000008  Initial Size   40/     8
   Slice Size      00000000.0000A000                    Released        0/     0
   Next free Slice 00000000.00000008

   Item                                Base              End          Length
   PFN Database                  FFFFFFFE.00000000  FFFFFFFE.00050000  00050000
```

The CLUE MEMORY/GH/FULL command displays data structures that describe huge pages.

4. SDA> CLUE MEMORY/LAYOUT
System Virtual Address Space Layout:

Item	Base	End	Length
System Virtual Base Address	FFFFFFFE.00000000		
PFN Database	FFFFFFFE.00000000	FFFFFFFE.00050000	00050000
Permanent Mapping of System L1PT	FFFFFFFE.00050000	FFFFFFFE.00052000	00002000
Global Page Table (GPT)	FFFFFFFE.00052000	FFFFFFFE.00063608	00011608
Lock ID Table	FFFFFFFF.7FFD0000	FFFFFFFF.80000000	00030000
Execlet Code Region	FFFFFFFF.80000000	FFFFFFFF.80400000	00400000
Resident Image Code Region	FFFFFFFF.80400000	FFFFFFFF.80C00000	00800000
System Header	FFFFFFFF.80D00000	FFFFFFFF.80D0A000	0000A000
Error Log Allocation Buffers	FFFFFFFF.80D0A000	FFFFFFFF.80D0C000	00002000
Nonpaged Pool (initial size)	FFFFFFFF.80D0E000	FFFFFFFF.80ECA000	001BC000
Nonpaged Pool Expansion Area	FFFFFFFF.80ECA000	FFFFFFFF.815BC000	006F2000
Execlet Data Region	FFFFFFFF.80C00000	FFFFFFFF.80D00000	00100000
Fork Buffers Secondary to Primary	FFFFFFFF.82982000	FFFFFFFF.82984000	00002000
Erase Pattern Buffer Page	FFFFFFFF.82990000	FFFFFFFF.82992000	00002000
63 Balance Slots - 3 pages each	FFFFFFFF.815C0000	FFFFFFFF.8173A000	0017A000
Paged Pool	FFFFFFFF.8173A000	FFFFFFFF.81820000	000E6000
System Control Block (SCB)	FFFFFFFF.81820000	FFFFFFFF.81828000	00008000
Restart Parameter Block (HWRPB)	FFFFFFFF.8186E000	FFFFFFFF.81872000	00004000
Erase Pattern Page Table Page	FFFFFFFF.82992000	FFFFFFFF.82994000	00002000
Posix Cloning Parent Page Mapping	FFFFFFFF.829D0000	FFFFFFFF.829D2000	00002000
Posix Cloning Child Page Mapping	FFFFFFFF.829D2000	FFFFFFFF.829D4000	00002000
Swapper Process Kernel Stack	FFFFFFFF.82A8C000	FFFFFFFF.82A8E000	00002000
Swapper Map	FFFFFFFF.82AA2000	FFFFFFFF.82AA8000	00006000
Idle Loop’s Mapping of Zero Pages	FFFFFFFF.82A8E000	FFFFFFFF.82A90000	00002000
PrimCPU Machine Check Logout Area	FFFFFFFF.8296A000	FFFFFFFF.8296C000	00002000
PrimCPU Sys Context Kernel Stack	FFFFFFFF.82966000	FFFFFFFF.82968000	00002000
Tape Mount Verification Buffer	FFFFFFFF.81824000	FFFFFFFF.81828000	00004000
Mount Verification Buffer	FFFFFFFF.82980000	FFFFFFFF.82982000	00002000
Demand Zero Optimization Page	FFFFFFFF.82C60000	FFFFFFFF.82C62000	00002000
Executive Mode Data Page	FFFFFFFF.82C62000	FFFFFFFF.82C64000	00002000
System Space Expansion Region	FFFFFFFF.84000000	FFFFFFFF.FFDF0000	7BDF0000
System Page Table Window	FFFFFFFF.FFDF0000	FFFFFFFF.FFFF0000	00200000
N/A Space	FFFFFFFF.FFFF0000	FFFFFFFF.FFFFFFFF	00010000

The CLUE MEMORY/LAYOUT command decodes and displays the system virtual address space layout.

5. SDA> CLUE MEMORY/LOOKASIDE

```
Non-Paged Dynamic Storage Pool - Lookaside List Queue Information:
------------------------------------------------------------------
Listhead Addr: FFFFFFFF.80C50400   Size:   64   Status: Valid, 11 elements
Listhead Addr: FFFFFFFF.80C50408   Size:  128   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50410   Size:  192   Status: Valid, 29 elements
Listhead Addr: FFFFFFFF.80C50418   Size:  256   Status: Valid, 3 elements
Listhead Addr: FFFFFFFF.80C50420   Size:  320   Status: Valid, 7 elements
Listhead Addr: FFFFFFFF.80C50428   Size:  384   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50430   Size:  448   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50438   Size:  512   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50440   Size:  576   Status: Valid, 6 elements
Listhead Addr: FFFFFFFF.80C50448   Size:  640   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50450   Size:  704   Status: Valid, 5 elements
Listhead Addr: FFFFFFFF.80C50458   Size:  768   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50460   Size:  832   Status: Valid, empty
Listhead Addr: FFFFFFFF.80C50468   Size:  896   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50470   Size:  960   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50478   Size: 1024   Status: Valid, 6 elements
Listhead Addr: FFFFFFFF.80C50480   Size: 1088   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50488   Size: 1152   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50490   Size: 1216   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50498   Size: 1280   Status: Valid, 2 elements
Listhead Addr: FFFFFFFF.80C504A0   Size: 1344   Status: Valid, 2 elements
Listhead Addr: FFFFFFFF.80C504A8   Size: 1408   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504B0   Size: 1472   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504B8   Size: 1536   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504C0   Size: 1600   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504C8   Size: 1664   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504D0   Size: 1728   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504D8   Size: 1792   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504E0   Size: 1856   Status: Valid, empty
Listhead Addr: FFFFFFFF.80C504E8   Size: 1920   Status: Valid, empty
Listhead Addr: FFFFFFFF.80C504F0   Size: 1984   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C504F8   Size: 2048   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50500   Size: 2112   Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C50508   Size: 2176   Status: Valid, 15 elements
Listhead Addr: FFFFFFFF.80C50510   Size: 2240   Status: Valid, empty
Listhead Addr: FFFFFFFF.80C50518   Size: 2304   Status: Valid, 1 element
   .
   .
   .
Total free space:  00016440 (hex)    91200 (dec)  bytes
```

The CLUE MEMORY/LOOKASIDE command summarizes the state of nonpageable lookaside lists. For each list, an indication of whether the queue is well formed is given. If a queue is not well formed or is invalid, messages indicating what is wrong with the queue are displayed. This command is analogous to the SDA command VALIDATE QUEUE.

These messages can also appear frequently when the VALIDATE QUEUE command is used within an SDA session that is analyzing a running system. In a running system, the composition of a queue can change while the command is tracing its links, thus producing an error message.

6.
```
SDA> CLUE MEMORY/STATISTIC
Memory Management Statistics:
-----------------------------
Pagefaults:                              Non-Paged Pool:
Total Page Faults              32181     Successful Exp Attempts            0
Total Page Reads               13017     Unsuccessful Exp Attempts          0
I/O's to read Pages             6131     Expansion Failures                 0
Modified Pages Written          1984     Failed Pages Accumulator           0
I/O's to write Mod Pages          31     Total Alloc Requests            3357
Demand Zero Faults             10068     Failed Alloc Requests              0
Global Valid Faults             6191     Paged Pool:
Modified Faults                 5724     Total Failures                     0
Read Faults                        0     Failed Pages Accumulator           0
Execute Faults                  1834     Total Alloc Requests            1633
                                         Failed Alloc Requests              0

Direct I/O                     13619     Cur Mapped Gbl Sections          391
Buffered I/O                   72046     Max Mapped Gbl Sections          392
Split I/O                        875     Cur Mapped Gbl Pages            7236
Hits                           14595     Max Mapped Gbl Pages            7257
Logical Name Transl           207730     Maximum Processes                 21
Dead Page Table Scans              0     Sched Zero Pages Created           0

Distributed Lock Manager:          Local         Incoming         Outgoing
$ENQ New Lock Requests             77626                0                0
$ENQ Conversion Requests          104843                0                0
$DEQ Dequeue Requests              77395                0                0
Blocking ASTs                         12                0                0
Directory Functions                                     0                0
Deadlock Messages                                       0                0

$ENQ Requests that Wait          136     Deadlock Searches Performed        2
$ENQ Requests not Queued           5     Deadlocks Found                    0

File System Cache:     Current SYSGEN Param        Hits      Misses Hitrate
File Header Cache     (ACP_HDRCACHE  = 126)        4753        1265   78.9%
Storage Bitmap Cache  (ACP_MAPCACHE  =  31)          11           6   64.7%
Directory Data Cache  (ACP_DIRCACHE  = 126)       12174         534   95.7%
Directory LRU         (ACP_DINDXCACHE=  31)       11158         175   98.4%
FID Cache             (ACP_FIDCACHE  =  64)          95           2   97.9%
Extent Cache          (ACP_EXTCACHE  =  64)         116           4   96.6%
Quota Cache           (ACP_QUOCACHE  =  65)           0           0    0.0%

Volume Synch Locks               341     Window Turns                      60
Volume Synch Locks Wait            0     Currently Open Files             313
Dir/File Synch Locks           19681     Total Count of OPENs            3038
Dir/file Synch Locks Wait         73     Total Count of ERASE QIOs          8
Access Locks                       0
Free Space Cache Wait             17

Global Pagefile Quota           1426     GBLPAGFIL (SYSGEN) Limit        1664
```

The CLUE MEMORY/STATISTIC command displays systemwide performance data such as page fault, I/O, pool, lock manager, MSCP, and file system cache.

CLUE PROCESS

Displays process-related information from the current process context.

Format

CLUE PROCESS [/qualifier[,...]]

Parameters

None.

Qualifiers

/BUFFER [/ALL]
Displays the buffer objects for the current process. If the /ALL qualifier is specified, then the buffer objects for all processes (that is, all existing buffer objects) are displayed.

/LAYOUT
Displays the process P1 virtual address space layout.

/LOGICAL
Displays the process logical names and equivalence names, if they can be accessed.

/RECALL
Displays the DCL recall buffer, if it can be accessed.

Description

The CLUE PROCESS command displays process-related information from the current process context. Much of this information is in pageable address space and thus may not be present in a dump file.

Examples

1. SDA> CLUE PROCESS/LOGICAL

```
Process Logical Names:
----------------------
   "SYS$OUTPUT" = "_CLAWS$LTA5004:"
   "SYS$OUTPUT" = "_CLAWS$LTA5004:"
   "SYS$DISK" = "WORK1:"
   "BACKUP_FILE" = "_$65$DUA6"
   "SYS$PUTMSG" = "...À...À.."
   "SYS$COMMAND" = "_CLAWS$LTA5004:"
   "TAPE_LOGICAL_NAME" = "_$1$MUA3:"
   "TT" = "LTA5004:"
   "SYS$INPUT" = "_$65$DUA6:"
   "SYS$INPUT" = "_CLAWS$LTA5004:"
   "SYS$ERROR" = "21C00303.LOG"
   "SYS$ERROR" = "_CLAWS$LTA5004:"
   "ERROR_FILE" = "_$65$DUA6"
```

   The CLUE PROCESS/LOGICAL command displays logical names for each running process.

2.
```
SDA> CLUE PROCESS/RECALL
Process DCL Recall Buffer:
--------------------------
Index  Command
  1    ana/sys
  2    @login
  3    mc sysman io auto /log
  4    show device d
  5    sea <.x>*.lis clue$
  6    tpu <.x>*0914.lis
  7    sh log *hsj*
  8    xd <.x>.lis
  9    mc ess$ladcp show serv
 10    tpu clue_cmd.cld
 11    ana/sys
```

The CLUE PROCESS/RECALL command displays a listing of the DCL commands that have been executed most recently.

CLUE STACK

Identifies and displays the current stack. Use the SDA command SHOW STACK to display and decode the whole stack for the more common bugcheck types.

Format

CLUE STACK

Parameters

None.

Qualifiers

None.

Description

The CLUE STACK command identifies and displays the current stack together with the upper and lower stack limits. In case of a FATALEXCPT, INVEXCEPTN, SSRVEXCEPT, UNXSIGNAL, or PGFIPLHI bugcheck, CLUE STACK tries to decode the whole stack.

Examples

1.
```
SDA> CLUE STACK
Stack Decoder:
--------------
Normal Process Kernel Stack:
Stack Pointer         FFFFFFFF.7FF91D58
Stack Limits (low)    FFFFFFFF.7FF90000
             (high)   FFFFFFFF.7FF92000
```

CLUE STACK identifies and displays the current stack together with the upper and lower stack limits.

2.
```
SDA> CLUE STACK
Stack Decoder:
--------------
Normal Process Kernel Stack:
Stack Pointer         00000000.7FFA1C98
Stack Limits (low)    00000000.7FFA0000
             (high)   00000000.7FFA2000

SSRVEXCEPT Stack:
-----------------
Stack Pointer  SP =>  00000000.7FFA1C98

Information saved by Bugcheck:
a(Signal Array)       00000000.7FFA1C98  00000000.00000000

EXE$EXCPTN[E] Temporary Storage:
EXE$EXCPTN[E] Stack Frame:
PV                    00000000.7FFA1CA0  FFFFFFFF.829CF010  EXE$EXCPTN
        Entry Point                      FFFFFFFF.82A21000  EXE$EXCPTN_C
return PC             00000000.7FFA1CA8  FFFFFFFF.82A2059C  SYS$CALL_HANDL_C+0002C
saved R2              00000000.7FFA1CB0  00000000.00000000
saved FP              00000000.7FFA1CB8  00000000.7FFA1CD0
```

```
SYS$CALL_HANDL Temporary Storage:
                      00000000.7FFA1CC0  FFFFFFFF.829CEDA8  SYS$CALL_HANDL
                      00000000.7FFA1CC8  00000000.00000000
SYS$CALL_HANDL Stack Frame:
PV                    00000000.7FFA1CD0  FFFFFFFF.829CEDA8  SYS$CALL_HANDL
        Entry Point                      FFFFFFFF.82A20570  SYS$CALL_HANDL_C
                      00000000.7FFA1CD8  00000000.00000000
return PC             00000000.7FFA1CE0  FFFFFFFF.82A1E930  CHF_REI+000DC
saved FP              00000000.7FFA1CE8  00000000.7FFA1F40

Fixed Exception Context Area:
Linkage Pointer       00000000.7FFA1CF0  FFFFFFFF.80C63780  EXCEPTION_MON_NPRW+06D80
a(Signal Array)       00000000.7FFA1CF8  00000000.7FFA1EB8
a(Mechanism Array)    00000000.7FFA1D00  00000000.7FFA1D40
a(Exception Frame)    00000000.7FFA1D08  00000000.7FFA1F00
Exception FP          00000000.7FFA1D10  00000000.7FFA1F40
Unwind SP             00000000.7FFA1D18  00000000.00000000
Reinvokable FP        00000000.7FFA1D20  00000000.00000000
Unwind Target         00000000.7FFA1D28  00000000.00020000  SYS$K_VERSION_04
#Sig Args/Byte Cnt    00000000.7FFA1D30  00000005.00000250  BUG$_NETRCVPKT
a(Msg)/Final Status   00000000.7FFA1D38  829CE050.000008F8  BUG$_SEQ_NUM_OVF

Mechanism Array:
Flags/Arguments       00000000.7FFA1D40  00000000.0000002C
a(Establisher FP)     00000000.7FFA1D48  00000000.7AFFBAD0
reserved/Depth        00000000.7FFA1D50  FFFFFFFF.FFFFFFFD
a(Handler Data)       00000000.7FFA1D58  00000000.00000000
a(Exception Frame)    00000000.7FFA1D60  00000000.7FFA1F00
a(Signal Array)       00000000.7FFA1D68  00000000.7FFA1EB8
saved R0              00000000.7FFA1D70  00000000.00020000  SYS$K_VERSION_04
saved R1              00000000.7FFA1D78  00000000.00000000
saved R16             00000000.7FFA1D80  00000000.00020004  UCB$M_NI_PRM_MLT+00004
saved R17             00000000.7FFA1D88  00000000.00010050  SYS$K_VERSION_16+00010
saved R18             00000000.7FFA1D90  FFFFFFFF.FFFFFFFF
saved R19             00000000.7FFA1D98  00000000.00000000
saved R20             00000000.7FFA1DA0  00000000.7FFA1F50
saved R21             00000000.7FFA1DA8  00000000.00000000
saved R22             00000000.7FFA1DB0  00000000.00010050  SYS$K_VERSION_16+00010
saved R23             00000000.7FFA1DB8  00000000.00000000
saved R24             00000000.7FFA1DC0  00000000.00010051  SYS$K_VERSION_16+00011
saved R25             00000000.7FFA1DC8  00000000.00000000
saved R26             00000000.7FFA1DD0  FFFFFFFF.8010ACA4  AMAC$EMUL_CALL_NATIVE_C+000A4
saved R27             00000000.7FFA1DD8  00000000.00010050  SYS$K_VERSION_16+00010
saved R28             00000000.7FFA1DE0  00000000.00000000
FP Regs not valid     [...............]
a(Signal64 Array)     00000000.7FFA1EA0  00000000.7FFA1ED0
SP Align = 10(hex)    [...............]

Signal Array:
Arguments             00000000.7FFA1EB8           00000005
Condition             00000000.7FFA1EBC           0000000C
Argument #2           00000000.7FFA1EC0           00010000  LDRIMG$M_NPAGED_LOAD
Argument #3           00000000.7FFA1EC4           00000000
Argument #4           00000000.7FFA1EC8           00030078  SYS$K_VERSION_01+00078
Argument #5           00000000.7FFA1ECC           00000003

64-bit Signal Array:
Arguments             00000000.7FFA1ED0  00002604.00000005
Condition             00000000.7FFA1ED8  00000000.0000000C
Argument #2           00000000.7FFA1EE0  00000000.00010000  LDRIMG$M_NPAGED_LOAD
Argument #3           00000000.7FFA1EE8  00000000.00000000
Argument #4           00000000.7FFA1EF0  00000000.00030078  SYS$K_VERSION_01+00078
Argument #5           00000000.7FFA1EF8  00000000.00000003
```

```
Interrupt/Exception Frame:
saved R2              00000000.7FFA1F00  00000000.00000003
saved R3              00000000.7FFA1F08  FFFFFFFF.80C63460  EXCEPTION_MON_NPRW+06A60
saved R4              00000000.7FFA1F10  FFFFFFFF.80D12740  PCB
saved R5              00000000.7FFA1F18  00000000.000000C8
saved R6              00000000.7FFA1F20  00000000.00030038  SYS$K_VERSION_01+00038
saved R7              00000000.7FFA1F28  00000000.7FFA1FC0
saved PC              00000000.7FFA1F30  00000000.00030078  SYS$K_VERSION_01+00078
saved PS              00000000.7FFA1F38  00000000.00000003  IPL INT CURR PREV
SP Align = 00(hex)    [...............]                     00  0  Kern User

Stack Frame:
PV                    00000000.7FFA1F40  00000000.00010050  SYS$K_VERSION_16+00010
        Entry Point                      00000000.00030060  SYS$K_VERSION_01+00060
                      00000000.7FFA1F48  00000000.00010000  LDRIMG$M_NPAGED_LOAD
return PC             00000000.7FFA1F50  FFFFFFFF.8010ACA4  AMAC$EMUL_CALL_NATIVE_C+000A4
saved FP              00000000.7FFA1F58  00000000.7FFA1F70

Stack (not decoded):
                      00000000.7FFA1F60  00000000.00000001
                      00000000.7FFA1F68  FFFFFFFF.800EE81C  RM_STD$DIRCACHE_BLKAST_C+005AC

Stack Frame:
PV                    00000000.7FFA1F70  FFFFFFFF.80C6EBA0  EXE$CMKRNL
        Entry Point                      FFFFFFFF.800EE6C0  EXE$CMKRNL_C
                      00000000.7FFA1F78  00000000.829CEDE8  EXE$SIGTORET
                      00000000.7FFA1F80  00010050.00000002
                      00000000.7FFA1F88  00000000.00020000  SYS$K_VERSION_04
                      00000000.7FFA1F90  00000000.00030000  SYS$K_VERSION_01
return PC             00000000.7FFA1F98  FFFFFFFF.800A4D64  __RELEASE_LDBL_EXEC_SERVICE+00284
saved R2              00000000.7FFA1FA0  00000000.00000003
saved R4              00000000.7FFA1FA8  FFFFFFFF.80D12740  PCB
saved R13             00000000.7FFA1FB0  00000000.00010000  LDRIMG$M_NPAGED_LOAD
saved FP              00000000.7FFA1FB8  00000000.7AFFBAD0

Interrupt/Exception Frame:
saved R2              00000000.7FFA1FC0  00000000.7FFCF880  MMG$IMGHDRBUF+00080
saved R3              00000000.7FFA1FC8  00000000.7B0E9851
saved R4              00000000.7FFA1FD0  00000000.7FFCF818  MMG$IMGHDRBUF+00018
saved R5              00000000.7FFA1FD8  00000000.7FFCF938  MMG$IMGHDRBUF+00138
saved R6              00000000.7FFA1FE0  00000000.7FFAC9F0
saved R7              00000000.7FFA1FE8  00000000.7FFAC9F0
saved PC              00000000.7FFA1FF0  FFFFFFFF.80000140  SYS$CLREF_C
saved PS              00000000.7FFA1FF8  00000000.0000001B  IPL INT CURR PREV
SP Align = 00(hex)    [...............]                     00  0  User User
```

CLUE STACK displays and decodes the current stack if it is one of the more popular and known bugcheck types. In this case, CLUE STACK trys to decode the whole INVEXCEPTN stack.

CLUE VCC

Displays virtual I/O cache-related information.

Format

CLUE VCC [/qualifier[,...]]

Parameters

None.

Qualifiers

/CACHE
Decodes and displays the cache lines that are used to correlate the file virtual block numbers (VBNs) with the memory used for caching. Note that the cache itself is not dumped in a selective dump. Use of this qualifier with a selective dump produces the following message:

```
%CLUE-I-VCCNOCAC, Cache space not dumped because DUMPSTYLE is selective
```

/LIMBO
Walks through the limbo queue (LRU order) and displays information for the cached file header control blocks (FCBs).

/STATISTIC
Displays statistical and performance information related to the virtual I/O cache.

/VOLUME
Decodes and displays the cache volume control blocks (CVCB).

Examples

1.
```
SDA> CLUE VCC/STATISTIC
Virtual I/O Cache Statistics:
-----------------------------
Cache State       pak,on,img,data,enabled
Cache Flags       on,protocol_only
Cache Data Area   80855200

Total Size (pages)             400     Total Size (MBytes)              3.1 MB
Free Size (pages)                0     Free Size (MBytes)               0.0 MB
Read I/O Count               34243     Read I/O Bypassing Cache        3149
Read Hit Count               15910     Read Hit Rate                    46.4%
Write I/O Count               4040     Write I/O Bypassing Cache        856
IOpost PID Action Rtns       40829     IOpost Physical I/O Count         28
IOpost Virtual I/O Count         0     IOpost Logical I/O Count           7
Read I/O past File HWM         124     Cache Id Mismatches               44
Count of Cache Block Hits      170     Files Retained                   100

Cache Line LRU    82B11220 82B11620    Oldest Cache Line Time     00001B6E
Limbo LRU Queue   80A97E3C 80A98B3C    Oldest Limbo Queue Time    00001B6F
Cache VCB Queue   8094DE80 809AA000    System Uptime (seconds)    00001BB0
```

2. SDA> CLUE VCC/VOLUME

```
Virtual I/O Cache - Cache VCB Queue:
-----------------------------------
CacheVCB RealVCB   LockID      IRP Queue     CID  LKSB Ocnt State
-------- -------- -------- ---------------- ---- ---- ---- --------------
8094DE80 80A7E440 020007B2 8094DEBC 8094DEBC 0000 0001 0002 on
809F3FC0 809F97C0 0100022D 809F3FFC 809F3FFC 0000 0001 0002 on
809D0240 809F7A40 01000227 809D027C 809D027C 0000 0001 0002 on
80978B80 809F6C00 01000221 80978BBC 80978BBC 0000 0001 0002 on
809AA000 809A9780 01000005 809AA83C 809AA03C 0007 0001 0002 on
```

3. SDA> CLUE VCC/LIMBO

```
Virtual I/O Cache - Limbo Queue:
-------------------------------
  CFCB     CVCB     FCB      CFCB   IOerrors   FID (hex)
-------- -------- -------- -Status- -------- -------------
80A97DC0 809AA000 80A45100 00000200 00000000 (076B,0001,00)
80A4E440 809AA000 809CD040 00000200 00000000 (0767,0001,00)
80A63640 809AA000 809FAE80 00000200 00000000 (0138,0001,00)
80AA2540 80978B80 80A48140 00000200 00000000 (0AA5,0014,00)
80A45600 809AA000 80A3AC00 00000200 00000000 (0C50,0001,00)
80A085C0 809AA000 809FA140 00000200 00000000 (0C51,0001,00)
80A69800 809AA000 809FBA00 00000200 00000000 (0C52,0001,00)
80951000 809AA000 80A3F140 00000200 00000000 (0C53,0001,00)
80A3E580 809AA000 80A11A40 00000200 00000000 (0C54,0001,00)
80A67F80 809AA000 80978F00 00000200 00000000 (0C55,0001,00)
809D30C0 809AA000 809F4CC0 00000200 00000000 (0C56,0001,00)
809D4B80 809AA000 8093E540 00000200 00000000 (0C57,0001,00)
[......]
80A81600 809AA000 8094B2C0 00000200 00000000 (0C5D,0001,00)
80AA3FC0 809AA000 80A2DEC0 00000200 00000000 (07EA,000A,00)
80A98AC0 809AA000 8093C640 00000200 00000000 (0C63,0001,00)
```

4. SDA> CLUE VCC/CACHE

```
Virtual I/O Cache - Cache Lines:
-------------------------------
   CL       VA       CVCB     CFCB     FCB      CFCB   IOerrors   FID (hex)
-------- -------- -------- -------- -------- -Status- -------- ------------
82B11200 82880000 809D0240 809D7000 80A01100 00000200 00000000 (006E,0003,00)
82B15740 82AAA000 809AA000 80A07A00 80A24240 00000000 00000000 (0765,0001,00)
82B14EC0 82A66000 809AA000 80A45600 80A3AC00 00000200 00000000 (0C50,0001,00)
82B12640 82922000 809D0240 809D7000 80A01100 00000200 00000000 (006E,0003,00)
82B123C0 8290E000 809AA000 80A45600 80A3AC00 00000200 00000000 (0C50,0001,00)
82B13380 8298C000 809D0240 809D7000 80A01100 00000200 00000000 (006E,0003,00)
82B15A40 82AC2000 809AA000 80A45600 80A3AC00 00000200 00000000 (0C50,0001,00)
82B15F40 82AEA000 809D0240 809D7000 80A01100 00000200 00000000 (006E,0003,00)
82B12AC0 82946000 809D0240 809D7000 80A01100 00000200 00000000 (006E,0003,00)
82B12900 82938000 809D0240 809D7000 80A01100 00000200 00000000 (006E,0003,00)
82B10280 82804000 809AA000 80A45600 80A3AC00 00000200 00000000 (0C50,0001,00)
82B122C0 82906000 809AA000 80A1AC00 80A48000 00000000 00000000 (0164,0001,00)
82B14700 82A28000 809AA000 809FFEC0 809F8DC0 00000004 00000000 (07B8,0001,00)
82B11400 82890000 809AA000 80A113C0 80A11840 00000000 00000000 (00AF,0001,00)
[......]
82B11380 8288C000 809AA000 809DA0C0 809C99C0 00002000 00000000 (00AB,0001,00)
82B130C0 82976000 809AA000 809DA0C0 809C99C0 00002000 00000000 (00AB,0001,00)
82B11600 828A0000 809AA000 809DA0C0 809C99C0 00002000 00000000 (00AB,0001,00)
```

CLUE XQP

Displays XQP-related information.

Format

CLUE XQP [/qualifier[,...]]

Parameters

None.

Qualifiers

/ACTIVE [/FULL]
Displays all active XQP processes.

/AQB
Displays any current I/O request packets (IRPs) waiting at the interlocked queue.

/BFRD=index
Displays the buffer descriptor (BFRD) referenced by the index specified. The index is identical to the hash value.

/BFRL=index
Displays the buffer lock block descriptor (BFRL) referenced by the index specified. The index is identical to the hash value.

/BUFFER=(n,m) [/FULL]
Displays the BFRDs for a given pool. Specify either 0, 1, 2 or 3, or a combination of these in the parameter list.

/CACHE_HEADER
Displays the block buffer cache header.

/FCB=address [/FULL]
Displays all file header control blocks (FCBs) with a nonzero DIRINDX for a given volume. If no address is specified, the current volume of the current process is used.

The address specified can also be either a valid volume control block (VCB), unit control block (UCB), or window control block (WCB) address.

/FILE=address
Decodes and displays file header (FCB), window (WCB), and cache information for a given file. The file can be identified by either its FCB or WCB address.

/GLOBAL
Displays the global XQP area for a given process.

/LBN_HASH=lbn
Calculates and displays the hash value for a given logical block number (LBN).

/LIMBO
Searches through the limbo queue and displays FCB information from available, but unused file headers.

/LOCK=lockbasis
Displays all file system serialization, arbitration, and cache locks found for the specified lockbasis.

/THREAD=n
Displays the XQP thread area for a given process. The specified thread number is checked for validity. If no thread number is specified, the current thread is displayed. If no current thread, but only one single thread is in use, then that thread is displayed. If more than one thread exists or an invalid thread number is specified, then a list of currently used threads is displayed.

/VALIDATE=(n,m)
Performs certain validation checks on the block buffer cache to detect corruption. Specify 1, 2, 3, 4, or a combination of these in the parameter list. If an inconsistency is found, a minimal error message is displayed. If you add the /FULL qualifier, additional information is displayed.

Description

The CLUE XQP command displays XQP information. XQP is part of the I/O subsystem.

Examples

1.
```
SDA> CLUE XQP/CACHE_HEADER
Block Buffer Cache Header:
-------------------------
Cache_Header   8437DF90   BFRcnt         000005D2   FreeBFRL       843916A0
Bufbase        8439B400   BFRDbase       8437E080   BFRLbase       8438F7E0
Bufsize        000BA400   LBNhashtbl     84398390   BFRLhashtbl    84399BC8
Realsize       000D78A0   LBNhashcnt     0000060E   BFRLhashcnt    0000060E

Pool           #0         #1          #2          #3
Pool_LRU       8437E5C0   84385F40    84387E90    8438EEB0
               8437F400   84385D60    8438AC80    8438EE20
Pool_WAITQ     8437DFE0   8437DFE8    8437DFF0    8437DFF8
               8437DFE0   8437DFE8    8437DFF0    8437DFF8
Waitcnt        00000000   00000000    00000000    00000000
Poolavail      00000094   00000252    00000251    00000094
Poolcnt        00000095   00000254    00000254    00000095

AmbigQFL       00000000   Process_Hits   00000000   Cache_Serial   00000000
AmbigQBL       00000000   Valid_Hits     00000000   Cache_Stalls   00000000
Disk_Reads     00000000   Invalid_Hits   00000000   Buffer_Stalls  00000000
Disk_Writes    00000000   Misses         00000000
```

The SDA command CLUE XQP/CACHE_HEADER displays the block buffer cache header.

2.
```
SDA> CLUE XQP/VALIDATE=(1,4)
Searching BFRD Array for possible Corruption...
Searching Lock Basis Hashtable for possible Corruption...
```

In this example, executing the CLUE XQP/VALIDATE=1,4 command indicated that no corruption was detected in either the BFRD Array or the Lock Basis Hashtable.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z