OpenVMS Alpha System Dump Analyzer Utility Manual

December 1995

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 AXP System Dump Analyzer Utility Manual*, Version 6.1
Software Version:	OpenVMS Alpha Version 7.0

Digital Equipment Corporation
Maynard, Massachusetts

December 1995

6635

This document 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.

Associated Documents

For additional information, refer to the following documents:

- *OpenVMS Alpha Version 7.0 Upgrade and Installation Manual*
- *OpenVMS Calling Standard*
- *OpenVMS System Manager's Manual*
- *OpenVMS Programming Interfaces: Calling a System Routine*
- *OpenVMS Alpha Device Support: Developer's Guide*
- *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 10 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.

The following conventions are used to identify information specific to OpenVMS Alpha or to OpenVMS VAX:

	The Alpha icon denotes the beginning of information specific to OpenVMS Alpha.
	The VAX icon denotes the beginning of information specific to OpenVMS VAX.
♦	The diamond symbol denotes the end of a section of information specific to OpenVMS Alpha or to OpenVMS VAX.

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 VMScluster 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*.) 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).

In a physical memory dump, the DUMPSTYLE system parameter can be set to either 0 or 2. Each value provides a full dump; the value of 0 yields minimal console output while the value of 2 provides 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 either 1 or 3 to specify a dump of selected virtual address spaces. Each value provides a selective dump; the value of 1 yields minimal console output while the value of 3 provides 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–1 compares full and selective style dump files.

Table SDA–1 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 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*.)

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 by 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

%SDA-I-READSYM, reading symbol table  SYS$COMMON:[SYS$LDR]REQSYSDEF.STB;1
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–2.

- **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–2 SDA Operators

Operator	Action
Unary Operators	
#	Performs a logical NOT of the expression.
+	Makes the value of the expression positive.
–	Makes the value of the expression negative.
@	Evaluates the following expression as a virtual address, then uses the contents of that address as value.
^Q	When used with the unary operator @, it specifies the size of field to be used as a virtual address is a quadword [1].
^L	When used with the unary operator @, it specifies the size of field to be used as a virtual address is a longword[2].
^W	When used with the unary operator @, it specifies the size of field to be used as a virtual address is a word[3].
^B	When used with the unary operator @, it specifies the size of field to be used as a virtual address is a byte[4].
G	Adds FFFFFFFF 80000000_{16} to the value of the expression[5].
H	Adds $7FFE0000_{16}$ to the value of the expression[6].

[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.

[5]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 $80000D40_{16}$.

[6]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.

(continued on next page)

Table SDA–2 (Cont.) SDA Operators

Operator	Action
Unary Operators	

I	Fills the leading digits of the following hexadecimal number with hex value of F. For example:
	`SDA> eval i80000000`
	`Hex = FFFFFFFF80000000 Decimal = --2147483648 G`
	`SYS$PUBLIC_VECTORS_NPRO`

Binary Operators	
+	Addition
–	Subtraction
*	Multiplication
&	Logical AND
|	Logical OR
\	Logical XOR
/	Division[7]
@	Arithmetic shifting

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

[7]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–3. For compatibility with OpenVMS VAX, the modules' file types have been renamed to .STB.

Table SDA–3 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–4 lists symbols that SDA defines automatically on initialization.

Table SDA–4 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	$7FFE0000_{16}$, a base address in P1 space
KSP	Kernel stack pointer
PC	Program counter
PS	Processor status
PTBR	Page table base register
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–5 are set for that CPU.

Table SDA–5 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–6 are defined for that CPU.

Table SDA–6 SDA Symbols Defined by SET PROCESS Command

ARB	Address of access rights block
JIB	Address of job information 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–6 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.

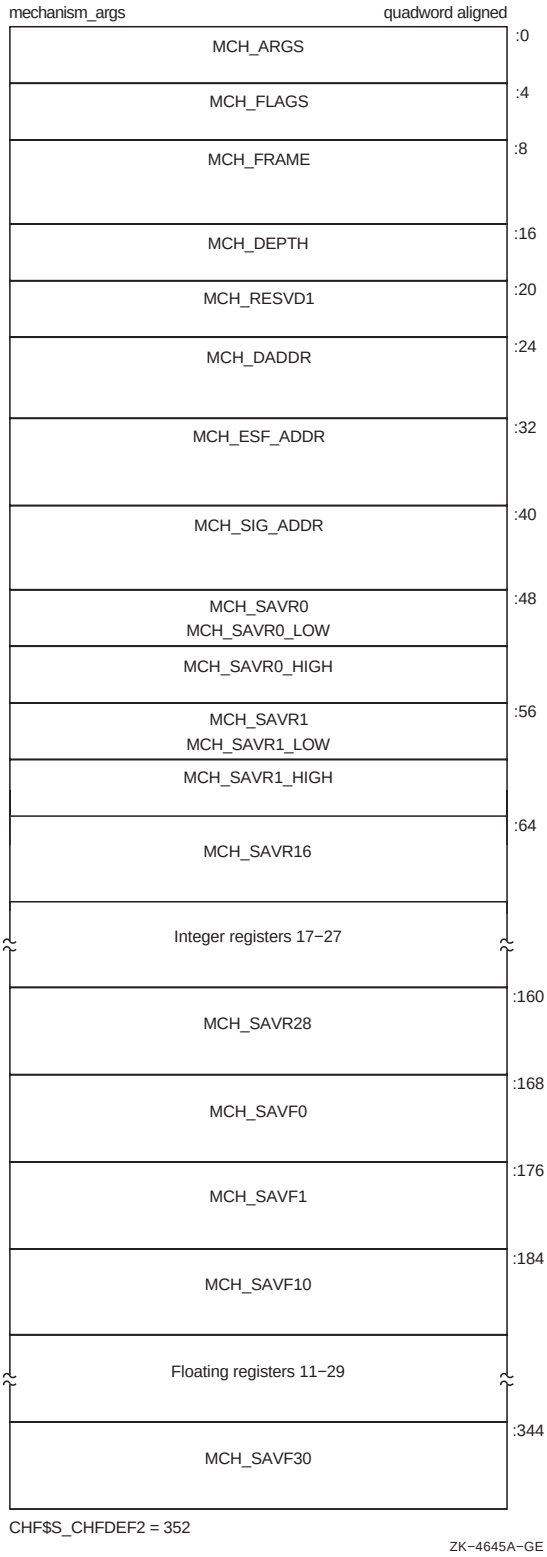
When the system fails, the operating system reports the approximate cause of the system failure on the console terminal. SDA displays a similar message when you issue a SHOW CRASH command. For instance, for a fatal exception, SDA can display one of these messages:

```
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 $2B_{16}$. These quadwords are used by the procedures that search for a condition handler and report exceptions.

Figure SDA–1 Mechanism Array

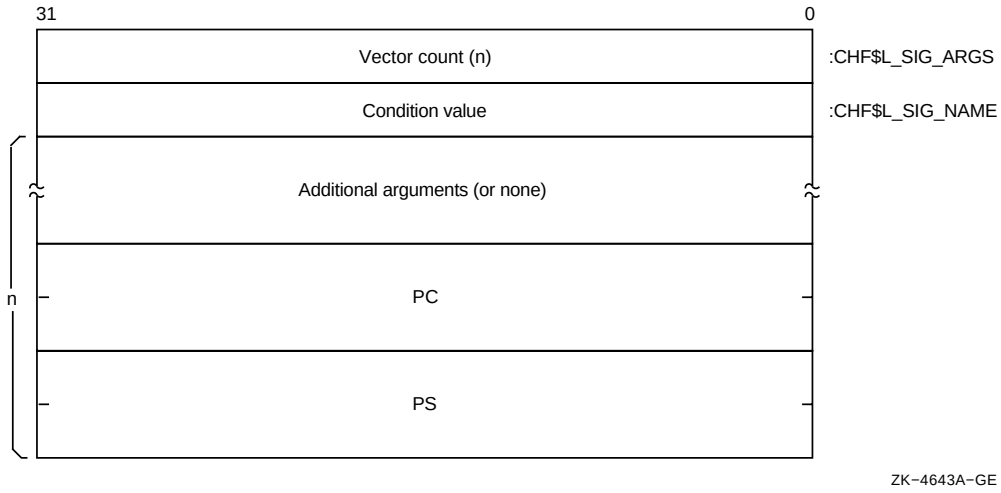


Symbolic offsets into the mechanism array are defined as follows. The SDA SHOW STACK command identifies the elements of the mechanism array on the stack using these symbols.

Offset	Meaning
CHF$IS_MCH_ARGS	Number of quadwords that follow. In a mechanism array, this value is always $2B_{16}$.
CHF$IS_MCH_FLAGS	Flag bits for related argument mechanism information.
CHF$IS_MCH_FRAME	Address of the FP (frame pointer) of the establisher's call frame.
CHF$IS_MCH_DEPTH	Depth of the OpenVMS Alpha search for a condition handler.
CHF$IS_MCH_DADDR	Address of the handler data quadword, if the exception handler data field is present.
CHF$IS_MCH_ESF_ADDR	Address of the exception stack frame (see Figure SDA–3).
CHF$IS_MCH_SIG_ADDR	Address of the signal array (see Figure SDA–2).
CHF$IS_MCH_SAVRnn	Contents of the saved integer registers at the time of the exception. The following registers are saved: R0, R1, and R16 to R28 inclusive.
CHF$IS_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.

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. The SDA SHOW STACK command uses the CHF$ symbols listed in the figure to identify the signal array on the stack.

Figure SDA–2 Signal Array

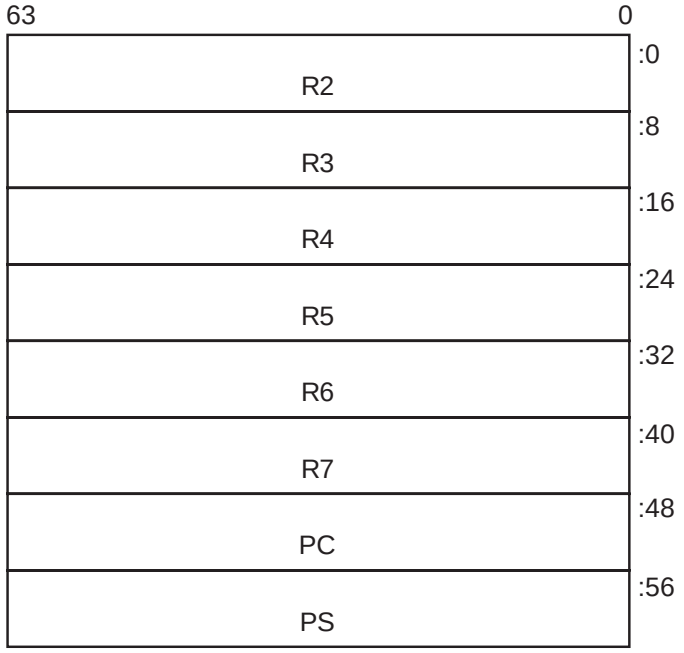


For access violations, the signal array is set up as follows:

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.

Figure SDA–3 illustrates the exception stack frame, which comprises all quadwords.

Figure SDA–3 Exception Stack Frame



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

Table SDA–7 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 array, and exception stack frame previously described.

```
OpenVMS Alpha System dump analyzer

Dump taken on 14-FEB-1995 16:39:37.79
SSRVEXCEPT, Unexpected system service exception
...analyzing a selective memory dump...

SDA> SHOW CRASH
Time of system crash: 14-FEB-1995 16:39:37.79

Version of system: OpenVMS Alpha Operating System, Version V7.0

System Version Major ID/Minor ID: 3/0

VMScluster node: ISFLM1, a DEC 3000 Model 500

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
Exception Frame:
----------------
        R2  = FFFFFFFF 8006F878  SCH$INIT_C+00674
        R3  = 00000000 00000000
        R4  = FFFFFFFF 805D7800
        R5  = 00000000 7FFA9CB0
        R6  = 00000000 7FF95E40
        R7  = FFFFFFFF 81F77E70  EXE$CRE_MIN_PROCESS+00430
        PC  = FFFFFFFF 81F6CDB4  EXE$CRE_MIN_PROCESS_C+00A14
        PS  = 00000000 00000000

        FFFFFFFF 81F6CDA4:      JSR             R26,(R26)
        FFFFFFFF 81F6CDA8:      LDQ             R27,#XFDE0(R13)
        FFFFFFFF 81F6CDAC:      LDL             R5,#X0238(R27)
        FFFFFFFF 81F6CDB0:      BEQ             R5,#X000002
   PC =>FFFFFFFF 81F6CDB4:      LDL             R26,(R5)
        FFFFFFFF 81F6CDB8:      BNE             R26,#X00000B
        FFFFFFFF 81F6CDBC:      LDQ             R25,#XFE00(R13)
        FFFFFFFF 81F6CDC0:      LDL             R5,#X0D18(R25)
        FFFFFFFF 81F6CDC4:      BEQ             R5,#X000002

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

%SYSTEM-F-PAGRDERR, page read error, reason mask=00, virtual address=7FFA8001,
  PC=81F6CDB4, PSL=00000000

Saved Scratch Registers in Mechanism Array
------------------------------------------
R0  = 00000000 00000002   R1  = 00000000 00000000   R16 = 00000000 00000000
R17 = 0000FFFE 00007C04   R18 = 00000000 00000000   R19 = FFFFFFFF 800650F0
R20 = 10000000 00000000   R21 = 00000000 00000000   R22 = 00000000 00000400
R23 = 00000000 2014002B   R24 = 00000000 00000000   R25 = 00000000 00000000
R26 = 00000000 00000002   R27 = 00000000 7FFF0000   R28 = FFFFFFFF 81F6CDA8

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

Process currently executing on this CPU: JOB_CONTROL

Current IPL: 0  (decimal)

CPU database address:  8052E000

General registers:
```

```
R0  = 00000000 00000444 R1  = 00000000 7FF95C28 R2  = FFFFFFFF 8006F878
R3  = 00000000 00000000 R4  = FFFFFFFF 805D7800 R5  = 00000000 7FFA9CB0
R6  = 00000000 7FF95E40 R7  = FFFFFFFF 81F77E70 R8  = 00000000 000203E8
R9  = 00000000 7FF59640 R10 = 00000000 7FF58608 R11 = 00000000 7FFD00A8
R12 = 00000000 00000000 R13 = FFFFFFFF 80778C10 R14 = 00000000 7FF57118
R15 = FFFFFFFF 80403400 R16 = 00000000 000003C4 R17 = 00000000 7FF95AC0
R18 = 00000000 00000168 R19 = FFFFFFFF 800650F0 R20 = 10000000 00000000
R21 = 00000000 00000000 R22 = FFFFFFFF 80778000 R23 = 00000000 7FF96000
R24 = 00000000 7FFF0024 AI  = 00000000 00000002 RA  = FFFFFFFF 81F43D4C
PV  = FFFFFFFF 80778C10 R28 = 00000000 000005AC FP  = 00000000 7FF95A20
PC  = FFFFFFFF 81F44254 PS  = 18000000 00000000


Processor Internal Registers:

ASN  = 00000000 0000000C                  ASTSR/ASTEN =          0000000F
IPL  =          00000000 PCBB = 00000000 0124A080 PRBR = FFFFFFFF 8052E000
PTBR = 00000000 00000A6D SCBB = 00000000 000001A5 SISR = 00000000 00000000

        KSP    = 00000000 7FF95A18
        ESP    = 00000000 7FF56AE0
        SSP    = 00000000 7FFA2000
        USP    = 00000000 7FF4A890

               No spinlocks currently owned by CPU 00

SDA> SHOW STACK
Current Operating Stack (KERNEL):
                        7FF959F8  18000000 00000000
                        7FF95A00  FFFFFFFF 80778A70  EXE$SET_PAGES_READ_ONLY+00630
                        7FF95A08  00000000 80778B38  EXE$SIGTORET
                        7FF95A10  00000000 7FF95AC0
                SP =>   7FF95A18  00000000 7FF95C28
                        7FF95A20  FFFFFFFF 80778C10  EXE$EXCPTN
                        7FF95A28  FFFFFFFF 81F43D4C  EXCEPTION_PRO+01D4C
                        7FF95A30  FFFFFFFF 81F77D20  EXE$CRE_MIN_PROCESS+002E0
                        7FF95A38  00000000 7FF95A50
                        7FF95A40  FFFFFFFF 80778AF8  EXE$SET_PAGES_READ_ONLY+006B8
                        7FF95A48  FFFFFFFF 8006F878  SCH$INIT_C+00674
                        7FF95A50  FFFFFFFF 80778AF8  EXE$SET_PAGES_READ_ONLY+006B8
                        7FF95A58  00000000 00000000
                        7FF95A60  FFFFFFFF 81F42790  EXE$CONTSIGNAL_C+001B0
                        7FF95A68  00000000 7FF95CC0
                        7FF95A70  FFFFFFFF 80428540  EXE$ACVIOLAT
                        7FF95A78  00000000 7FF95C28
                        7FF95A80  00000000 7FF95AC0
                        7FF95A88  00000000 7FF95C80
                        7FF95A90  00000000 7FF95CC0
                        7FF95A98  00000000 00000000
                        7FF95AA0  00000000 00000000
                        7FF95AA8  FFFFFFFF 8041E840  EXE$KP_DEALLOCATE_KPB
                        7FF95AB0  00000005 00000250  UCB$T_MSGDATA+00034
                        7FF95AB8  80778000 000008F8  UCB$M_VALID+000F8
CHF$IS_MCH_ARGS         7FF95AC0  00000000 0000002B
CHF$PH_MCH_FRAME        7FF95AC8  00000000 7FF56AE0
CHF$IS_MCH_DEPTH        7FF95AD0  FFFFFFFF FFFFFFFD
CHF$PH_MCH_DADDR        7FF95AD8  FFFFFFFF 8045C3E0  SYS$DKDRIVER_NPRW+001E0
CHF$PH_MCH_ESF_ADDR     7FF95AE0  00000000 7FF95C80
CHF$PH_MCH_SIG_ADDR     7FF95AE8  00000000 7FF95C28
CHF$IH_MCH_SAVR0        7FF95AF0  00000000 00000002
CHF$IH_MCH_SAVR1        7FF95AF8  00000000 00000000
CHF$IH_MCH_SAVR16       7FF95B00  00000000 00000000
CHF$IH_MCH_SAVR17       7FF95B08  0000FFFE 00007C04
CHF$IH_MCH_SAVR18       7FF95B10  00000000 00000000
CHF$IH_MCH_SAVR19       7FF95B18  FFFFFFFF 800650F0  PROCESS_MANAGEMENT_NPRO+050F0
```

```
CHF$IH_MCH_SAVR20       7FF95B20  10000000 00000000
CHF$IH_MCH_SAVR21       7FF95B28  00000000 00000000
CHF$IH_MCH_SAVR22       7FF95B30  00000000 00000400  IRP$M_MBXIO
CHF$IH_MCH_SAVR23       7FF95B38  00000000 2014002B
CHF$IH_MCH_SAVR24       7FF95B40  00000000 00000000
CHF$IH_MCH_SAVR25       7FF95B48  00000000 00000000
CHF$IH_MCH_SAVR26       7FF95B50  00000000 00000002
CHF$IH_MCH_SAVR27       7FF95B58  00000000 7FFF0000  CTL$GL_NMIOCH
CHF$IH_MCH_SAVR28       7FF95B60  FFFFFFFF 81F6CDA8  EXE$CRE_MIN_PROCESS_C+00A08
                        7FF95B68  00000000 00000000
                        7FF95B70  00000000 00000000
                        7FF95B78  00000000 00000000
                        7FF95B80  00000000 00000000
                        7FF95B88  00000000 00000000
                        7FF95B90  00000000 00000000
                        7FF95B98  00000000 00000000
                        7FF95BA0  00000000 00000000
                        7FF95BA8  00000000 00000000
                        7FF95BB0  00000000 00000000
                        7FF95BB8  00000000 00000000
                        7FF95BC0  00000000 00000000
                        7FF95BC8  00000000 00000000
                        7FF95BD0  00000000 00000000
                        7FF95BD8  00000000 00000000
                        7FF95BE0  00000000 00000000
                        7FF95BE8  00000000 00000000
                        7FF95BF0  00000000 00000000
                        7FF95BF8  00000000 00000000
                        7FF95C00  00000000 00000000
                        7FF95C08  00000000 00000000
                        7FF95C10  00000000 00000000
                        7FF95C18  00000000 00000000
                        7FF95C20  00000000 00000000
CHF$L_SIG_ARGS          7FF95C28  00000444 00000005  UCB$M_SHD_SEQCMD_HERE+00044
CHF$L_SIG_ARG1          7FF95C30  7FFA8001 00000000
                        7FF95C38  00000000 81F6CDB4  EXE$CRE_MIN_PROCESS_C+00A14
                        7FF95C40  00000002 00000001
                        7FF95C48  00000000 00000444  UCB$M_SHD_SEQCMD_HERE+00044
                        7FF95C50  00000000 00000000
                        7FF95C58  FFFFFFFF 81F6CDB4  EXE$CRE_MIN_PROCESS_C+00A14
                        7FF95C60  00000008 00000000
                        7FF95C68  00000000 00000000
                        7FF95C70  00000008 00000000
                        7FF95C78  00000000 7FFA8001
INTSTK$Q_R2             7FF95C80  FFFFFFFF 8006F878  SCH$INIT_C+00674
INTSTK$Q_R3             7FF95C88  00000000 00000000
INTSTK$Q_R4             7FF95C90  FFFFFFFF 805D7800
INTSTK$Q_R5             7FF95C98  00000000 7FFA9CB0
INTSTK$Q_R6             7FF95CA0  00000000 7FF95E40
INTSTK$Q_R7             7FF95CA8  FFFFFFFF 81F77E70  EXE$CRE_MIN_PROCESS+00430
INTSTK$Q_PC             7FF95CB0  FFFFFFFF 81F6CDB4  EXE$CRE_MIN_PROCESS_C+00A14
INTSTK$Q_PS             7FF95CB8  00000000 00000000
Prev SP (7FF95CC0) ==>  7FF95CC0  FFFFFFFF 81F77E70  EXE$CRE_MIN_PROCESS+00430
                        7FF95CC8  00000000 00000000
                        7FF95CD0  00000000 00000005
                        7FF95CD8  00000000 00000000
                        7FF95CE0  FFFFFFFF FFFF42D4
                        7FF95CE8  00000000 0000001F
                        7FF95CF0  FFFF8000 00000000
   .
   .
   .
```

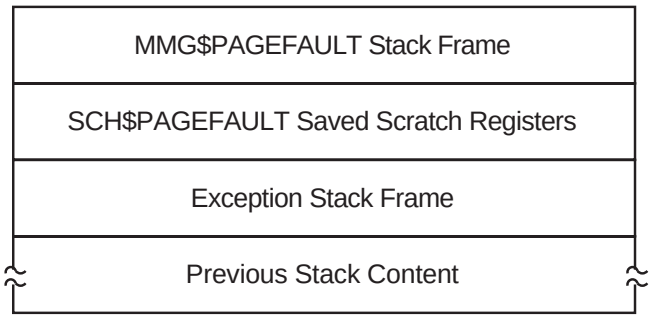
6.2.2 Illegal Page Faults

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–4.

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



ZK–6787A–GE

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–3)
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–7. 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 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 7.0 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 FFFFFFFF00000000
>>> 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] filespec | /SYSTEM]

/SYMBOL=system-symbol-table

Command Parameter

filespec

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

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:

```
SDA 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
/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.

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.

/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
```

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.

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 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
READ
REPEAT
SEARCH
SET CPU
SET FETCH
SET LOG
SET OUTPUT
SET PROCESS
SET RMS
SET SIGN_EXTEND
SHOW CALL_FRAME
SHOW CLUSTER
SHOW CONNECTIONS
SHOW CPU
SHOW CRASH
SHOW DEVICE
SHOW EXECUTIVE
SHOW HEADER
SHOW LAN
SHOW LOCK
SHOW MACHINE_CHECK
SHOW PAGE_TABLE
SHOW PFN_DATA
SHOW POOL
SHOW PORTS
SHOW PROCESS
SHOW RESOURCE
SHOW RMS
SHOW RSPID
SHOW SPINLOCKS
SHOW STACK
SHOW SUMMARY
SHOW SYMBOL
SPAWN
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):
                         7FF95D00  00000000 0000000B
                         7FF95D08  FFFFFFFF 804395C8  MMG$TBI_DATA_64+000B8
                         7FF95D10  00000000 00000000
                         7FF95D18  0000FE00 00007E04
                  SP =>  7FF95D20  00000000 00000800  IRP$M_EXTEND
                         7FF95D28  00000001 000002F7  UCB$B_PI_FKB+0000B
                         7FF95D30  FFFFFFFF 804395C8  MMG$TBI_DATA_64+000B8
                         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):
   .
   .
   .
```

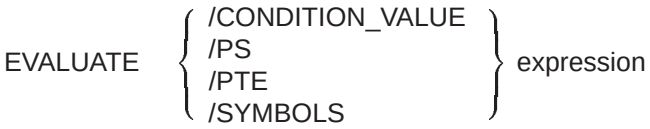
The first DEFINE/KEY command defines PF1 as a key that sets a command state GREEN. The trailing pair of quotation marks is required syntax, indicating that no command is to be executed when this key is pressed.

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



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.

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.

Examples

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

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

2.
```
SDA> EVALUATE 1
Hex = 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 = 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 = 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 $-\frac{1}{4}$ 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 ABCDFFEE
```

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

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 for full dumps only. 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 FFE00000
```

```
 3 3 2  2              2   1   1 1
 1 0 9  7              0   8   6 5               7 6           0
+-+-+--+--------------+-+-+---+-+---------------+-+-----------+-+
|1|1|03|     007F     |0|X| 00|0|      00       |X|    00     |0|
+-+-+--+--------------+-+-+---+-+---------------+-+-----------+-+
|                            00000000                          |
+--------------------------------------------------------------+
Demand Zero PTE: Owner = K, Read Prot = NONE, Write Prot = NONE, CPY = 3
```

   The EXAMINE/PTE command displays and formats the system page table entry at FFE00000.

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

Parameter

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

Qualifier

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

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:

```
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–3 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:

```
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$LDR]SYSDEF.STB
SDA>FORMAT G41F818
FFFFFFFF8041F818   UCB$L_FQFL                      8041F818 UCB
                   UCB$L_MB_MSGQFL
                   UCB$L_RQFL
                   UCB$W_MB_SEED
                   UCB$W_UNIT_SEED
FFFFFFFF8041F81C   UCB$L_FQBL                      8041F818 UCB
                   UCB$L_MB_MSGQBL
                   UCB$L_RQBL
FFFFFFFF8041F820   UCB$W_SIZE                          0110
FFFFFFFF8041F822   UCB$B_TYPE                        10
FFFFFFFF8041F823   UCB$B_FLCK                      2C
FFFFFFFF8041F824   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.

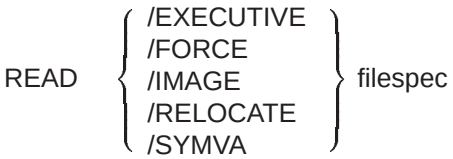
10.
```
SDA> MAP 7FC56000
Demand Zero Data Section                Base      End       Image Offset
LIBRTL                                  7FC56000  7FC57000  000E0000
```

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

READ

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

Format



Parameter

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

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.

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

/FORCE
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.

/IMAGE
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.

/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 for which the default symbol table (SDA$READ_DIR:SYS$BASE_IMAGE.EXE) provides no definition. In other words, the required global symbols are located in modules that have been compiled and linked separately from the executive. SDA extracts no local symbols from the object module.

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–3 lists the files that OpenVMS Alpha provides in SYS$LOADABLE_ IMAGES that define data structure offsets.

Table SDA–8 lists the files in SYS$LOADABLE_IMAGES that define global locations within executive images.

Table SDA–8 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
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

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

(continued on next page)

Table SDA–8 (Cont.) Modules Defining Global Locations Within Executive Image

File	Contents
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

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):

        000000007FF95CD0  FFFFFFFF 80430CE0  SCH$STATE_TO_COM+00040
        000000007FF95CD8  00000000 00000000
        000000007FF95CE0  FFFFFFFF 81E9CB04  LNM$SEARCH_ONE_C+000E4
        000000007FF95CE8  FFFFFFFF 8007A988  PROCESS_MANAGEMENT_NPRO+0E988
   SP =>000000007FF95CF0  00000000 00000000
        000000007FF95CF8  00000000 006080C1
        000000007FF95D00  FFFFFFFF 80501FDC
        000000007FF95D08  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):

               7FF95CD0  FFFFFFFF 80430CE0  SCH$FIND_NEXT_PROC
               7FF95CD8  00000000 00000000
               7FF95CE0  FFFFFFFF 81E9CB04  LNM$SEARCH_ONE_C+000E4
               7FF95CE8  FFFFFFFF 8007A988  SCH$INTERRUPT+00068
        SP =>  7FF95CF0  00000000 00000000
               7FF95CF8  00000000 006080C1
               7FF95D00  FFFFFFFF 80501FDC
               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.

/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.

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 800B81F0 to 800B86F0 in LONGWORD steps for 00060068...
Match at 800B8210
SDA>
```

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

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

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

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

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

SET CPU

Selects a processor to become the SDA current CPU.

Format

SET CPU cpu-id

Parameter

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

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

Qualifiers

None.

Description

When you invoke SDA to examine a system dump, the SDA current CPU context defaults to that of the processor that caused the system to fail. When analyzing a 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 FETCH

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

Format

SET FETCH [quad | long | word | byte]

Parameter

quad
Sets the default size to 8 bytes.

long
Sets the default size to 4 bytes.

word
Sets the default size to 2 bytes.

byte
Sets the default size to 1 byte.

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.

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, FFFFFFFFFF7FE000 : 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.

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 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.

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 | /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.

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

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

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

/SYSTEM
Specifies 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–9 lists those keywords that may be used as options.

Table SDA–9 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–9 (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, 0000000080400000: virtual data not in physical memory
```

This shows the SET SIGN_EXTEND command as OFF.

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

Parameters

None.

Qualifiers

/CSID=csid
Displays VMScluster system information for a specific VMScluster 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.

/NODE=name
Displays cluster information on a particular VMScluster member node which is specified by its SCS node name. This is mutually exclusive with the /CSID qualifier.

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

Description

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

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

The **VMScluster 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 VMScluster 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 VMScluster 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 VMScluster nodes.)

VMScluster 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 VMScluster 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
VMScluster data structures


            --- VMScluster 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

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

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

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
```

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) 805E7ED0 ---

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

Local Con. ID   FF120000    Datagrams sent         0    Message queue   805E7F00
Remote Con. ID  00000000    Datagrams rcvd         0    Send Credit Q.  805E7F08
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

   --- 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)

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                   ASTEN/ASTSR =          0000000E
IPL  =          0000000D PCBB = 00000000 03742080 PRBR = FFFFFFFF 805AE000
PTBR = 00000000 00000F34 SCBB = 00000000 00000500 SISR = 00000000 00000000

        KSP    = 00000000 7FF95A00
        ESP    = 00000000 7FF9A000
        SSP    = 00000000 7FFA04C0
        USP    = 00000000 7FE719F0


               Spinlocks currently owned by CPU 00

SCHED                              Address   80427880
Owner CPU ID       00000000        IPL       00000008
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

Parameters

None.

Qualifiers

None.

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=00000008, PC=8010D480,
PSL=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

            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

        FFFFFFFF80BB4A1C:      BLE             R0,#X000009
        FFFFFFFF80BB4A20:      BIS             R31,R1,R17
        FFFFFFFF80BB4A24:      ADDQ            R2,#X04,R16
        FFFFFFFF80BB4A28:      BIS             R31,R0,R25
  PC => FFFFFFFF80BB4A2C:      INSQUEL/D
        FFFFFFFF80BB4A30:      LDQ             R24,#X0078(R13)
        FFFFFFFF80BB4A34:      BIS             R31,R25,R0
        FFFFFFFF80BB4A38:      SUBL            R0,#X01,R0
        FFFFFFFF80BB4A3C:      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$DUA     F11XQP    SYS$DKDRIV 807735B0
      8000B2B8    RED$DUA       F11XQP    SYS$DKDRIV 807735B0
      80D08BA0    BIGTOP$DUA    F11XQP    SYS$DKDRIV 807735B0
      80D08AE0    TIMEIN$DUA    F11XQP    SYS$DKDRIV 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 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–8 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 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 7FF96000 00000000 7FF91C08 00000000 0000104B 040001C1 00000000  ....Á...K.........ù.....`ù.....  00000000
00001FFF 0000000D 00002000 00000000 00000000 7FA21AB0 00000000 7FF9C100  .Áù.....°.¢.......... ..........  00000020
0000F362 00000000 0D0D0D01 0000000D 0D040500 FCFFFFFF 03000000 8D810000  ..........ü.................hó..  00000040
65746365 74656420 00000000 534D5220 3454462D 43513558 0000000D 0000F362  hó......X5QC-FT4 RMS.... detecte  00000060
4145444E 41435326 00006B6F 69746964 6E6F6320 64696C61 766E6920 6B612064  d an invalid condition..&SCANDEA  00000080
00726F72 72652065 6C626174 20656761 70206461 6564206E 61635320 2C545044  DPT, Scan dead page table error.  000000A0
6F632065 636B6572 65666572 206B6F69 74636553 202C4745 4B464552 43455330  0SECREFNBG, Section reference co  000000C0
4E544E43 52485323 00000000 00000065 76697461 67656E20 746E6577 20746E75  unt went negative.......#SHRCNTN  000000E0
00000000 00020000 0004FFF9 00000000 6F63206i 72616873 204E0264 202C4745  EG, d.N shar. co....ù.........  00000100
⋮
Saved error log messages
------------------------
0004FFF9 0000040B 00000001 00000000 0000006C 80946B00 B094600C 00000000  .....`...h..l.............ù...  B0946000
1EBE0026 60030000 00000000 00000020 20544E54 53484308 00000000 00020000  .........CHSTNT  .........`&.Ç.  B0946020
30303320 43454417 00000287 0D0D0D00 3454467D 43513558 00D3D097 95B3EDA7  §í°.....X5QC-FT4.........DEC 3D0  B0946040
00000000 00010004 00000000 00000000 00000000 00303034 206C6564 6F4D2030  0 Model 400.....................  B0946060
20544E54 53484308 00000000 00020041 4B942454 4E545348 430A0000 000000F0  Ç.....CHSTNT$DKA........CHSTNT    B0946080
3454462D 43513558 00012020 41544C45 442D344B 35580000 00D00D00 00000020  ..........X5K4-DELTA  ..X5QC-FT4  B09460A0
00000000 00303034 206C6564 6F4D2030 30303320 43454412 00000002 00000000  .........DEC 3000 Model 400.....  B09460C0
4B942454 4E545348 430A012C 00000184 00000000 00010001 00000000 00000000  .....,...CHSTNT$DK  B09460E0
534D5650 58410000 00000000 00000000 00000000 00000000 00000000 00000041  A.....................AXPVMS  B0946100
⋮
```

ZK-6789A-GE

The SHOW HEADER command displays the contents of the dump file's header. Ellipses indicate hexadecimal information omitted from the display; two slashes (//) indicate an interruption in the ASCII 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.

/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.

/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.

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 12-FEB-1995 11:08:45 --

LAN flags:  00000002 LAN_INIT

LAN module version                1    First SVAPTE               FFE06960
LAN address               805636C0    Number of PTEs                    2
Number of stations                1    SVA of first page          81A58000
First LSB address         8057B240


            -- LAN CSMACD Network Management 12-FEB-1995 11:08:45 --

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 12-FEB-1995 11:08:45 --

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
NM flags:   00000000
```

```
                -- ESA Device Information 12-FEB-1995 11:08:45 --

LSB address                 8057B240    Active unit count                 3
ADP address                 80572600    IDB address                8054F900
Driver version     00000001 01050019    Driver code address        8046BDE0
Device1 version    00000000 00000000    Device1 code address       00000000
Device2 version    00000000 00000000    Device2 code address       00000000
LAN version        00000001 01050037    LAN code address           8046BFE0
Device name                 ES_LANCE    DLL type                     CSMACD
MOP ID                            39    MOP name                        SVA
HW version                  00000000    HW serial              Not supplied

Flags:  00000000
Char:   00000000
Status: 00000003 INITED,RUN


             -- ESA Device Information (cont) 12-FEB-1995 11:08:45 --

Fork block R3      00000000 00000000    Alt fork block R3  00000000 00000000
Fork pending flag           00000000    Receive ring size                16
Min receive buffers                9    Max receive buffers              17
Put rcv ptr/index           00000000    Get rcv ptr/index          0000000D
Put xmt ptr/index           8057BBB8    Get xmt ptr/index          8057BBB8
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    All multicast state             OFF
XEnts owned by dev                 0    Promiscuous mode                OFF
XEnts owned by host                4    Hardware mode              00000000
Controller mode               NORMAL    Hardware address   08-00-2B-1D-B7-58
Internal loopback                OFF    Physical address   AA-00-04-00-BD-FD
CRC generation mode               ON


             -- ESA Device Information (cont) 12-FEB-1995 11:08:45 --

DAT stage                   00000000    DAT xmt status     0000003C 003C0001
DAT number started                 1    DAT xmt complete     11-FEB 14:09:57
DAT number failed                  0    DAT rcv found                  None

Creation time                   None    Create count                      0
Deletion time                   None    Enable count                      0
Enabled time                    None    Fatal error count                 0
Disabled time                   None    Number of ports                   0

Last fork sched      12-FEB 11:08:44    Last fork time       12-FEB 11:08:44
Last receive         12-FEB 11:08:44    Last transmit        12-FEB 11:08:41
Last CRC error                  None    Last exc collision             None
Last length error               None    Last loss of carrier           None
Last USB error                  None    Last late collision            None
Last UUB error       12-FEB 10:19:25    Prev fatal error               None
Last fatal error                None


             -- ESA Device Information (cont) 12-FEB-1995 11:08:45 --

System buffer quota                0    Min 1st chain segment             0
Additional quota                   1    Min transmit length               0
Maximum quota                      8    Receive alignment                 0
Device dependent longword   00000000    Receive buffer size            1518
# restarts pending                0    Dev xmt header size               0

NMgmt advised buffer count         0    Events logged                     0
EIB address                 00000000    NMgmt assigned adr             None
```

```
                -- ESA Queue Information 12-FEB-1995 11:08:45 --

Control hold queue      8057B428  Status:  Valid, empty
Control request queue   8057B430  Status:  Valid, empty
Control pending queue   8057B438  Status:  Valid, empty
Transmit request queue  8057B420  Status:  Valid, empty
Transmit pending queue  8057B440  Status:  Valid, empty
Receive buffer queue    8057B470  Status:  Valid, 10 elements
Receive pending queue   8057B448  Status:  Valid, empty
Post process queue      8057B450  Status:  Valid, empty
Delay queue             8057B458  Status:  Valid, empty
Auto restart queue      8057B460  Status:  Valid, empty
Netwrk mgmt hold queue  8057B468  Status:  Valid, empty

          -- ESA Multicast Address Information 12-FEB-1995 11:08:45 --

AB-00-00-04-00-00


                  -- ESA Unit Summary 12-FEB-1995 11:08:45 --

UCB      UCB Addr  Fmt   Value           Client     State
---      --------  ---   -----           ------  -----------
ESA0     8054F980
ESA2     80584E00  Eth   60-03           DECnet  0017 STRTN,LEN,UNIQ,STRTD
ESA3     805A2980  Eth   60-04           LAT     0015 STRTN,UNIQ,STRTD
ESA5     805A7780  Eth   80-41           LAST    0015 STRTN,UNIQ,STRTD


              -- ESA Counters Information 12-FEB-1995 11:08:45 --

Octets received             95212950    Octets sent                43030048
PDUs received                 684755    PDUs sent                    170284
Mcast octets received       78765974    Mcast octets sent            478125
Mcast PDUs received           581005    Mcast PDUs sent                6385
Unrec indiv dest PDUs              0    PDUs sent, deferred           13511
Unrec mcast dest PDUs              0    PDUs sent, one coll           10136
Data overruns                      0    PDUs sent, mul coll           10554
Unavail station buffs              0    Excessive collisions              0
Last USB time                   None    Last exc collision             None
Unavail user buffers               2    Carrier check failure             0
Last UUB time       12-FEB 10:19:25     Last carrier failure           None
CRC errors                         0    Short circuit failure             0
Seconds since zeroed           75467    Open circuit failure              0
Station failures                   0    Transmits too long                0


           -- ESA Counters Information (cont) 12-FEB-1995 11:08:45 --

Last CRC time                   None    Late collisions                   0
Last CRC srcadr                 None    Last late collision            None
Alignment errors                   0    Coll detect chk fail              0
Frames too long                    0    Send data length err              0
Rcv data length err                0    Frame size errors                 0

Fatal error count                  0    Last fatal error               None
Restart failures                   0    Prev fatal error               None
Power failures                     0    Last error CSR             00000000
Transmit timeouts                  0    Fatal error code               None
Control timeouts                   0    Prev fatal error               None
Invalid length                     0
```

```
          -- ESA Counters Information (cont) 12-FEB-1995 11:08:45 --

Internal counters address   8057C0E0    Internal counter 1            134704
Internal counters size            28    Internal counter 2                 0
No work transmits                  0    Internal counter 3                 0
Buffer_Adr transmits               0    Internal counter 4                 0
SVAPTE/BOFF transmits              0    Internal counter 5                 0
Global page transmits              0    Internal counter 6                 0
Bad PTE transmits                  0    Internal counter 7                 0
Loopback sent                      0    Loopback failures                  0
System ID sent                   250    System ID failures                 0
ReqCounters sent                   0    ReqCounters failures               0


            -- ESA0 Template Unit Information 12-FEB-1995 11:08:45 --

LSB address                 8057B240    Error count                        0
VCIB address                00000000    Controller mode               NORMAL
Starter's PID               00000000    Internal loopback                OFF
Creator's PID               00000000    Access mode                   EXCLUSV
LAN medium                    CSMACD    Promiscuous mode                 OFF
Packet format               Ethernet    All multicast mode               OFF
Eth protocol type             00-00     Padding mode                      ON
802E protocol ID      00-00-00-00-00    Automatic restart                OFF
802.2 SAP                   00000000    Allow prom client                 ON
802.2 Group SAPs         A8,FA,0D,A7    Can change address               OFF
Maximum header size               0     802.2 service                    OFF
Device buffer size             1500     CRC generation mode               ON
Maximum buffer size            1500     Maintenance state                 ON
Hrdwre buffer quota               9     User transmit FC                  ON
Rcv buffer quota                  0     User receive FC                  OFF
Rcv buffs to queue                1     Default FC value                  00
Hardware address   08-00-2B-1D-B7-58    Physical address   FF-FF-FF-FF-FF-FF


          -- ESA2 60-03 (DECnet) Unit Information 12-FEB-1995 11:08:45 --

LSB address                 8057B240    Error count                        0
VCIB address                00000000    Controller mode               NORMAL
Starter's PID               0001000C    Internal loopback                OFF
Creator's PID               0001000C    Access mode                   EXCLUSV
LAN medium                    CSMACD    Promiscuous mode                 OFF
Packet format               Ethernet    All multicast mode               OFF
Eth protocol type             60-03     Padding mode                      ON
802E protocol ID      00-00-00-00-00    Automatic restart                OFF
802.2 SAP                   00000000    Allow prom client                 ON
802.2 Group SAPs         A8,FA,0D,A7    Can change address               OFF
Maximum header size              16     802.2 service                    OFF
Device buffer size             1500     CRC generation mode               ON
Maximum buffer size            1498     Maintenance state                 ON
Hrdwre buffer quota               9     User transmit FC                  ON
Rcv buffer quota              15040     User receive FC                  OFF
Rcv buffs to queue               10     Default FC value                  00
Hardware address   08-00-2B-1D-B7-58    Physical address   AA-00-04-00-BD-FD
```

```
        -- ESA2 60-03 (DECnet) Counters & Misc Info 12-FEB-1995 11:08:45 --

Last receive          12-FEB 11:08:40    Last transmit        12-FEB 11:08:39
Octets received             14227801    Octets sent                 41873076
PDUs received                  64065    PDUs sent                     121441
Mcast octets received        1576501    Mcast octets sent             302160
Mcast PDUs received            10077    Mcast PDUs sent                 5036
Unavail user buffer                2    Last start attempt              None
Last UUB time         12-FEB 10:19:25    Last start done      11-FEB 14:09:58
Multicast not enabled              0    Last start failed               None
User buff too small                0    Share UCB total quota              0

Receive IRP queue       80585070  Status:  Valid, 1 element
Shared users queue      80585060  Status:  Valid, empty
Receive pending queue   80585068  Status:  Valid, empty

       -- ESA2 60-03 (DECnet) Multicast Address Info 12-FEB-1995 11:08:45 --

Multicast address table, embedded:
  AB-00-00-04-00-00
```

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.

SHOW LOCK

Displays information about all lock management locks in the system, or about a specified lock.

Format

SHOW LOCK { lock-id | /ALL | /CACHED | /NAME=name }

Parameter

lock-id
Name of a specific lock.

Qualifiers

/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–10 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–10 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–10 (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:  00010001   PID:    00000000   Flags:   NOQUEUE SYNCSTS SYSTEM
Par. id:  00000000   Granted at    EX            CVTSYS
Sublocks:        1
LKB:      80D0B8A0
Resource:      5F535953 24535953    SYS$SYS_  Status:  NOQUOTA
 Length   16   00000000 4C774449    IDwL....
 Exec. mode    00000000 00000000    ........
 System        00000000 00000000    ........
Local copy
```

```
Lock id:  00010004   PID:    00000000   Flags:   CONVERT SYNCSTS CVTSYS
Par. id:  00000000   Granted at    CR
Sublocks:       16
LKB:      80D091A0   BLKAST
Resource:      4D567624 42313146    F11B$vVM  Status:  NOQUOTA
 Length   18   20204E41 4A353153    S15JAN
 Kernel mode   00000000 00002020      ......
 System        00000000 00000000    ........
Local copy


Lock id:  00280009   PID:    00000000   Flags:   VALBLK  CONVERT SYNCSTS
Par. id:  00000000   Granted at    CR            NOQUOTA CVTSYS
Sublocks:        0
LKB:      80CDA880
Resource:      52414B5F 24535953    SYS$_KAR  Status:  MSTCPY
 Length   17   30415544 24455441    ATE$DUA0
 Kernel mode   00000000 0000003A    :........
 System        00000000 00000000    ........
Master copy of lock 001C00F5 on system 000100A1
   .
   .
   .
SDA>  SHOW RESOURCE/LOCK=280009
Resource database
-----------------
Address of RSB:  80BD2150  Group grant mode:       CR
Parent RSB:      00000000  Conversion grant mode:  CR
Sub-RSB count:          0  BLKAST count:            0
Value block:   00000000 00000000 00000000 00000019   Seq. #:  0000002D
Resource:      52414B5F 24535953    SYS$_KAR
 Length   17   30415544 24455441    ATE$DUA0         CSID:  00000000
 Kernel mode   00000000 0000003A    :........
 System        00000000 00000000    ........

Granted queue (Lock ID / Gr mode):
     00DA1269  CR       00280009  CR       0094054D  CR
     00270B9F  CR       00D70BFE  CR       000D0F4F  CR
     000D1017  CR       00601418  CR       01131450  CR
     000F1964  CR       000200DF  CR

Conversion queue (Lock ID / Gr/Rq mode):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode):
     *** EMPTY QUEUE ***
```

This SDA session shows the output of the SHOW LOCK command for several locks. The SHOW RESOURCE command, executed for the last displayed lock, verifies that the lock is in the resource's granted queue, among many other locks given concurrent read (CR) access to the resource. (See Table SDA–17 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 [/qualifier[,...]] [range]

Parameter

range
Range of virtual addresses for which SDA is to display page table entries. You can express a range using the following 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

/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 process L1 page table.

/L2
Lists the process L2 page table.

/L3
Lists the process L3 page table.

/PT
Specifies page table space as the address range, as viewed in the context of the current process, or as viewed from system context if there is no current process.

/S0S1
Specifies S0 and S1 space as the address range.

/S2
Specifies S2 space as the address range.

/SPTW
Displays the contents of the system page table window. Level qualifiers are ignored.

/ALL
Lists both the global and system page tables.

Description

For each virtual address displayed by the SHOW PAGE_TABLE command, the first six columns of the listing provide the associated page table entry and describe its location, characteristics, and contents. SDA obtains this information from the system page table. Table SDA–11 describes the information displayed by the SHOW PAGE_TABLE command.

Table SDA–11 Virtual Page Information in the SHOW PAGE_TABLE Display

Value	Meaning
ADDRESS	System virtual address that marks the base of the virtual page.
SVAPTE	System virtual address of the page table entry that maps the virtual page. Equal values in the two SVAPTE columns indicates a valid link between physical and virtual address space.
PTE	Contents of the page table entry, a quadword that describes a system virtual page.
Type	Type of virtual page. There are the following eight types:

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.

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.
WRITE	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. The following codes are listed:

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.

GH	Contents of granularity hint bits.

If the virtual page has been mapped to a physical page, the last seven columns of the listing include information from the page frame number (PFN) database Otherwise, the section is left blank. Table SDA–12 describes the physical page information displayed by the SHOW PAGE_TABLE command.

Table SDA–12 Physical Page Information in the SHOW PAGE_TABLE Display

Category	Meaning
PAGTYP	Type of physical page. It is one of the following types:

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.
GPGTBL	Page is part of a global page table.
GBLWRT	Page is part of a global, writable section.
UNKNOWN	Unknown.

LOC — Location of the page within the system. It is one of the following eight types:

Location	Meaning
ACTIVE	Page is in a working set.
MDFYLST	Page is in the modified-page list.
FREELST	Page is in the free-page list.
BADLST	Page is in the bad-page list.
RELPEND	Release of the page is pending.
RDERROR	Page has had an error during an attempted read operation.
PAGEOUT	Page is being written into a paging file.
PAGEIN	Page is being brought into memory from a paging file.

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.

SVAPTE — System virtual address of the page table entry that maps the virtual page. Equal values in the two SVAPTE columns indicates a valid link between physical and virtual address space.

(continued on next page)

Table SDA–12 (Cont.) Physical Page Information in the SHOW PAGE_TABLE Display

Category	Meaning
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.

SDA indicates pages are inaccessible by displaying the following message:

```
-------- n NULL PAGES
```

Here, *n* indicates the number of inaccessible pages.

SHOW PFN_DATA

Displays information that is contained in the page lists and PFN database.

Format

SHOW PFN_DATA { [/qualifier] | [pfn] [;length] | [pfn] [:end-pfn] }

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.

/FREE
Displays the free-page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/MODIFIED
Displays the modified-page list. SDA precedes the list with a count of the pages it contains, its low limit, and its high limit.

/SYSTEM
Displays the entire PFN database in order by page frame number, starting at PFN 0000.

Description

For each page frame number it displays, the SHOW PFN_DATA command lists information used in translating physical page addresses to virtual page addresses. The display has two lines of information. Table SDA–13 shows the first line's fields; Table SDA–14 shows the second line's fields.

Table SDA–13 Page Frame Number Information—Line One Fields

Item	Contents
PFN	Page frame number.
DB ADDRESS	Address of PFN structure for this page.
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. It is one of the following eight types:

Location	Meaning
ACTIVE	Page is in a working set.
MDFYLST	Page is in the modified-page list.
FREELST	Page is in the free-page list.
BADLST	Page is in the bad-page list.
RELPEND	Release of the page is pending.
RDERROR	Page has had an error during an attempted read operation.
PAGEOUT	Page is being written into a paging file.
PAGEIN	Page is being brought into memory from a paging file.

(continued on next page)

Table SDA–13 (Cont.) Page Frame Number Information—Line One Fields

Item	Contents
FLAGS	Displays in text form the flags that are set in page state. Possible flags are:

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.

Table SDA–14 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.
Blank	
Blank	
Blank	
Blank	
REFCNT	Number of references being made to this page.
PAGETYP	Type of physical page. It is one of the following:

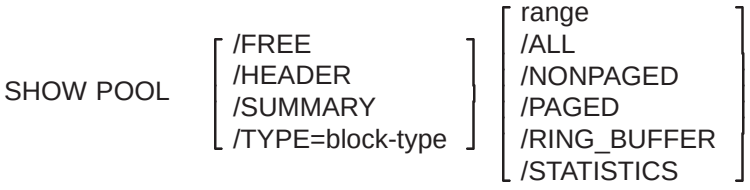
Page Type	Meaning
PROCESS	Page is part of process space.
SYSTEM	Page is part of system space.
GLOBAL	Page is part of a global section.
PPT(Ln)	Page is part of a process page table, where n is the page table level number.
GPGTBL	Page is part of a global page table.
GBLWRT	Page is part of a global, writable section.
UNKNOWN	Unknown.

Blank	

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



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.

/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.

/SUMMARY
Displays *only* an allocation summary for each specified region of pool.

/TYPE=block-type
Displays the blocks within the specified region or regions of pool that are of the indicated **block-type**. If SDA finds no blocks of that type in the pool region, it displays a blank screen, followed by an allocation summary of the region.

Description

The SHOW POOL command displays information about the contents of any specified region of pool in an 8-column format. 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. This 4-column table 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.

Examples

1.
```
SDA> SHOW POOL G0BADE00;260
Non-paged dynamic storage pool
------------------------------
                    Dump of blocks allocated from non-paged pool


CIMSG   80BADE00   144
                     001000DA 003C0090 0000A900 00036FF0 .o........<.....
                     D9B3001C 00000000 A0B5001D 35E60017 ...5............
                     41414141 00000600 65EA0004 00000600 .......e....AAAA
                     41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                     41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
   .
   .
   .
UNKNOWN 80BADE90   112
                     41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                     41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                     41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
                     41414141 41414141 41414141 41414141 AAAAAAAAAAAAAAAA
   .
   .
   .
CIDG    80BADED0   144
                     807708BB 003B0090 0004D7E0 000008F0 ..........;...w.
                     61616161 61616161 61616161 016CE87C ..l.aaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
   .
   .
   .
UNKNOWN 80BADF60    64
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
   .
   .
   .
CIDG    80BADFA0   144
                     807708BB 003B0090 0003FFC0 0004B1B0 ..........;...w.
                     61616161 61616161 61616161 016CE94C L.l.aaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
   .
   .
   .
UNKNOWN 80BAE030    48
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa
                     61616161 61616161 61616161 61616161 aaaaaaaaaaaaaaaa


Summary of non-paged pool contents

    3  UNKNOWN   =      176  (29%)
    2  CIDG      =      288  (48%)
    1  CIMSG     =      144  (24%)

Total space used = 608 out of 608 total bytes, 0 bytes left

Total space utilization = 100%
```

This example 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    8024FE00   528
                        802DC710 00380210 00000000 FFFFFF80 ..........8...-.
    LNM     80250010    96
                        8015B847 00400060 802D75A0 00000000 .....u-.`.@.G...
    LNM     80250070    48
                        8015B847 01400030 802500A0 802D7400 .t-...%.0.@.G...
    LNM     802500A0    96
                        8015B847 02400060 802DC170 80250070 p.%.p.-.`.@.G...
    LNM     80250100    48
                        8015B847 00400030 802DC510 802E1B60 `.....-.0.@.G...
       .
       .
       .
```

The SHOW POOL/PAGED/HEADER command displays only the name of each block allocated from paged pool, its starting address, its size, and the first 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 VMScluster 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 [/qualifier[,...]] [/ADDRESS=pcb-address | ALL | process-name | /INDEX=nn | /SYSTEM]

Parameters

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

process-name
Name of the process for which information is to be displayed. 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. 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.

/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). You can supply either of the following values for **nn**:

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

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

/INDEX=n
Displays the software and hardware context of the thread which is specified by the index of the software PCB into the system's PCB vector. Alternately, this value could be the process identification (PID or EPID) from which SDA extracts the correct index.

/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–10 for additional information.

/PAGE_TABLES [**range** | **/L1** | **/L2** | **/L3** | **/P0** | **/P1** | **/P2** | **/RDE=id**]

Displays the page tables of the process program (P0) and control (P1) regions, the process L1 and L2 page tables, both the P0 and P1 page tables (/L3), or, optionally, either the page table or the page table entries for a **range** of addresses.

The /RDE=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:n Displays the page table entries that correspond to the range of virtual addresses from *m* to *n*

m;n Displays the page table entries that correspond to a range of *n* pages, starting with page *m*

/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
Lists the information contained in the process section table (PST).

/RDE=id
Lists the information contained in the process region table for the specified region. If no region is specified, the entire table is displayed, along with the process-permanent regions.

/REGIONS
Displays the process region table entries for a particular process, or for a particular region if the qualifier is supplied with a region ID for a value. Is a synonym for the /RDE qualifier.

/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–9 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
Displays the working set list for the process.

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 the following information for each entry in the working set list:

Column	Contents
INDEX	Index into the working set list at which information for this entry can be found
ADDRESS	Virtual address of the page that this entry describes
STATUS	Three columns that list the following status information: • Location of the page in physical memory • Page status of VALID • Indication of whether the page is locked into the working set

When SDA locates one or more unused working set entries, it issues the following message:

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

In this message, *n* is the number (in decimal) of contiguous, unused entries.

The **process section table information** and **process section table** displays, also produced by the /PROCESS_SECTION_TABLE qualifier, list each entry in the process section table (PST) and display the offsets to, and the indices of, the first free entry and last used entry.

SDA displays the information listed in Table SDA–15 for each PST entry.

Table SDA–15 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	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.
CHANNEL	Address of the channel 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.
CLUSTER	Cluster size used when faulting pages into this process section.
REFCNT	Number of pages of this section that are currently mapped.
FLINK	Forward link. The pointer to the next entry in the PST list.
BLINK	Backward link. The pointer to the previous entry in the PST list.
FLAGS	Flags that describe the access that processes have to the process section.

The **P0 page table**, **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–11 and SDA–12).

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–16.

Table SDA–16 Process I/O Channel Information in the SHOW PROCESS Display

Information Displayed[1]	Type of Process
dcuu:	SDA displays this information for devices that are not file structured, such as terminals, and for processes that do not open files in the normal way.
dcuu:*filespec*	SDA displays this information only if you are examining a running system, and only if your process has enough privilege to translate the *file-id* into the *filespec*.
dcuu:(*file-id*)*filespec*	SDA displays this information only when you are examining a dump. The *filespec* corresponds to the *file-id* on the device listed. If you are examining a dump from your own system, the *filespec* is probably valid. If you are examining a dump from another system, the *filespec* is probably meaningless in the context of your system.
dcuu:(*file-id*)	The *file-id* no longer points to a valid *filespec*, as when you look at a dump from another system; or the process in which you are running SDA does not have enough privilege to translate the *file-id* into the corresponding *filespec*.

[1]This table uses the following 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    Buffered 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
AST's 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
```

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:          02040001    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  000000000000190A    Previous ASN     000000000000003F
Initial process priority        4    Delete pending count          0
# open files allowed left     100    Direct I/O count/limit      150/150
UIC                [00001,000200]    AST's remaining               97
Mutex count                     0    Buffered I/O count/limit      18/18
Abs time of last event   005D9941    BUFIO byte count/limit     32128/32128
AST's 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
AST's 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

  0001  FFFFFFFD FF7FC000     VALID PPT(L1) WSLOCK
  0002  FFFFFFFD FF000000     VALID PPT(L2) WSLOCK
  0003  FFFFFFFC 001FE000     VALID PPT(L3) WSLOCK
  0004  00000000 7FFA0000     VALID PROCESS MODIFIED WSLOCK
  0005  00000000 7FFF0000     VALID PROCESS WSLOCK
  0006  FFFFFFFF 81260000     VALID PHD     WSLOCK
Locked entries:
  0007  00000000 7B108000     VALID PROCESS WSLOCK
  0008  00000000 7B10A000     VALID PROCESS WSLOCK
Dynamic entries:
  0009  00000000 7B054000     VALID GLOBAL
  000A  00000000 7B0B0000     VALID GLOBAL
  000B  FFFFFFFC 001EC000     VALID PPT(L3) WSLOCK
  000C  00000000 7B0D0000     VALID GLOBAL
  000D  00000000 7B0C4000     VALID GLOBAL
  000E  00000000 7B0C0000     VALID GLOBAL
  000F  00000000 7FFA4000     VALID PROCESS
  0010  00000000 7FFD0000     VALID PROCESS
  0011  00000000 7FF96000     VALID PROCESS
  0012  00000000 7B0C6000     VALID GLOBAL
  0013  00000000 7B0DC000     VALID GLOBAL
  0014  00000000 7B0E4000     VALID GLOBAL
  0015  00000000 7B0E6000     VALID GLOBAL
  0016  00000000 7B0DE000     VALID GLOBAL
  0017  00000000 7FFAA000     VALID PROCESS
  0018  00000000 7B0E2000     VALID GLOBAL
  0019  00000000 7FFCE000     VALID PROCESS
  001A  00000000 7B0D2000     VALID GLOBAL
  001B  00000000 7B13E000     VALID PROCESS
  001C  00000000 7B140000     VALID PROCESS
  001D  00000000 7B0EA000     VALID GLOBAL
  001E  00000000 7B0CE000     VALID GLOBAL
  001F  00000000 7B068000     VALID GLOBAL
  0020  00000000 7B0CC000     VALID GLOBAL
  0021  00000000 7B07C000     VALID GLOBAL
  0022  00000000 7B07E000     VALID GLOBAL
  0023  00000000 7B084000     VALID GLOBAL
  0024  00000000 7B086000     VALID GLOBAL
  0025  00000000 7FFB8000     VALID PROCESS
  0026  00000000 7B144000     VALID PROCESS
  0027  FFFFFFFC 00000000     VALID PPT(L3)
  0028  00000000 7FF88000     VALID PROCESS
  0029  00000000 7FFBA000     VALID PROCESS

  ----  8 empty entries

  0032  00000000 7FF8A000     VALID PROCESS
```

```
  ----  6 empty entries

  0039  00000000 7B0D6000     VALID GLOBAL
  003A  00000000 7B0D8000     VALID GLOBAL

  ----  3 empty entries

  003E  00000000 7B0DA000     VALID GLOBAL

  ----  8 empty entries

  0047  00000000 7B066000     VALID GLOBAL
  0048  00000000 7B104000     VALID PROCESS
  0049  00000000 7B0B8000     VALID GLOBAL
  004A  00000000 7B07A000     VALID GLOBAL

  ----  11 empty entries

  0056  00000000 7B13A000     VALID PROCESS
  0057  00000000 7B13C000     VALID PROCESS

  ----  81 empty entries

  00A9  00000000 7FFEE000     VALID PROCESS
  00AA  00000000 7B142000     VALID PROCESS
     00AB  00000000 7FFB0000     VALID PROCESS
  00AC  00000000 7B0FE000     VALID PROCESS
  00AD  00000000 7B09E000     VALID PROCESS
  00AE  00000000 7B0A0000     VALID PROCESS
  00AF  00000000 7B0A2000     VALID PROCESS
  00B0  00000000 7B0A4000     VALID PROCESS
  00B1  00000000 7B100000     VALID PROCESS

  ----  18 empty entries

  00C4  00000000 7B138000     VALID PROCESS


Process section table information
---------------------------------

 Last entry allocated     0000
 First free entry         0000

P0 Space
--------

No pages allocated to this region

                       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
-------- -------- -------- -------- --------- ---------- ---------  --------

Total images = 0  Pages allocated = 0

                   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 P0 page table, the P1 page table, the P2 page table, 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, and /CHANNEL qualifiers, respectively.

3. SDA> SHOW PROCESS/PAGE_TABLES/ADDRESS=805E7980

```
P0 page table
-------------

ADDRESS   SVAPTE        PTE         TYPE  READ WRIT BITS GH PAGTYP    LOC
  BAK            REFCNT  SVAPTE   FLINK    BLINK

        -------- 8 NULL PAGES

00010000 80C08040 00000A2E 00160F09 VALID KESU NONE M-U- 00 PROCESS ACTIVE  0300
000000000000          1 80C08040 00000000 000000DC
00012000 80C08048 00000A2F 00160F09 VALID KESU NONE M-U- 00 PROCESS ACTIVE  0300
000000000000          1 80C08048 00000000 000000DD
00014000 80C08050 00000A30 00160F09 VALID KESU NONE MLU- 00 PROCESS ACTIVE  0300
000000000000          1 80C08050 00000000 000000C7
00016000 80C08058 00000A31 00060F09 VALID KESU NONE --U- 00 PROCESS ACTIVE  0300
000000000000          1 80C08058 00000000 000000DF

        -------- 4 NULL PAGES

00020000 80C08080 00000AA1 0016FF09 VALID KESU KESU M-U- 00 PROCESS ACTIVE  0300
000000000000          1 80C08080 00000000 000000F4

        -------- 7 NULL PAGES

00030000 80C080C0 00000A35 00060F01 VALID KESU NONE -LU- 00 PROCESS ACTIVE  0000
FFE200010000          1 80C080C0 00000000 000000C6
00032000 80C080C8 00000A36 00060F01 VALID KESU NONE --U- 00 PROCESS ACTIVE  0000
FFE200010000          1 80C080C8 00000000 000000E1
00034000 80C080D0 00000A37 00060F01 VALID KESU NONE --U- 00 PROCESS ACTIVE  0000
FFE200010000          1 80C080D0 00000000 000000E2
```

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 PA  BASE VA
-------- ------ -------- -------- -------- -------- -------- --------
805E4580 User   00000008 00000001 00010020 00000001 00020000 826BC000
805E7880 User   00000009 00000001 00010020 00000001 00020000 826BE000
8057AEC0 User   0000000A 00000001 00010020 00000001 00020000 826C0000
805E6EC0 User   0000000B 00000001 00010020 00000001 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 { /ALL | /CACHED | /LOCKID=lock-id | /NAME=resource-name }

Parameters

None.

Qualifiers

/ALL
Displays information from all resource blocks (RSBs) in the system. This is the default behavior of the SHOW RESOURCE command.

/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–17 for each resource in the system or for the specific resource associated with the specified **lock-id**.

Table SDA–17 Resource Information in the SHOW RESOURCE Display

Field	Contents
Address of RSB	Address of the resource block (RSB) that describes this resource.
Parent RSB	Address of the RSB that is the parent of this RSB. This field is 00000000 if the RSB itself is a parent block.
Sub-RSB count	Number of RSBs of which this RSB is the parent. This field is 0 if the RSB has no sub-RSBs.

(continued on next page)

Table SDA–17 (Cont.) Resource Information in the SHOW RESOURCE Display

Field	Contents
Group grant mode	Indication of the most restrictive mode in which a lock on this resource has been granted. This field can contain the following values (shown in order from the least restrictive mode to the most restrictive):

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

For information on conflicting and incompatible lock modes, see the *OpenVMS System Services Reference Manual*.

Conversion grant mode	Indication of the most restrictive lock mode to which a lock on this resource is waiting to be converted. This does not include the mode for which the lock at the head of the conversion queue is waiting.
BLKAST count	Number of locks on this resource that have requested a blocking AST.
Value block	Hexadecimal dump of the 16-byte block value block associated with this resource.
Sequence #	Sequence number associated with the resource's value block. If the number indicates that the value block is not valid, the words "Not valid" appear to the right of the number.
CSID	Cluster system identification number (CSID) of the node that owns the resource.
Resource	Dump of the name of this resource, as stored at the end of the RSB. The first two columns are the hexadecimal representation of the name, with the least significant byte represented by the rightmost two digits in the rightmost column. The third column contains the ASCII representation of the name, the least significant byte being represented by the leftmost character in the column. Periods in this column represent values that correspond to nonprinting ASCII characters.
Length	Length in bytes of the resource name.
Mode	Processor mode of the namespace in which this RSB resides.

(continued on next page)

Table SDA–17 (Cont.) Resource Information in the SHOW RESOURCE Display

Field	Contents
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.
Granted queue	List of locks on this resource that have been granted. For each lock in the list, SDA displays the number of the lock and the lock mode in which the lock was granted.
Conversion queue	List of locks waiting to be converted from one mode to another. For each lock in the list, SDA displays the number of the lock, the mode in which the lock was granted, and the mode to which the lock is to be converted.
Waiting queue	List of locks waiting to be granted. For each lock in the list, SDA displays the number of the lock and the mode requested for that lock.

Example

```
SDA>  SHOW RESOURCE
Resource database
-----------------
Address of RSB:  807F6120  Group grant mode:       NL
Parent RSB:      806EA180  Conversion grant mode:  NL
Sub-RSB count:          0  BLKAST count:            0
Value block:   806CE510 00000000 00000002 00000002   Seq. #:  00000008
Resource:      09ED7324 42313146    F11B$sí.
 Length   10   00000000 00000200    ........         CSID:  00020041
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........

Granted queue (Lock ID / Gr mode):
     006801AE  NL

Conversion queue (Lock ID / Gr/Rq mode):
     *** EMPTY QUEUE ***

Waiting queue (Lock ID / Rq mode):
     *** EMPTY QUEUE ***


Address of RSB:  807EB9E0  Group grant mode:       PW
Parent RSB:      00000000  Conversion grant mode:  EX
Sub-RSB count:          0  BLKAST count:            1
Value block:   00000000 00000003 00000000 0000FFF2   Seq. #:  0000027F Not valid
Resource:      32245F24 44414853    SHAD$_$2
 Length   16   3A31534A 44243435    54$DJS1:         CSID:  0002001A
 Kernel mode   00000000 00000000    ........
 System        00000000 00000000    ........
```

```
Granted queue (Lock ID / Gr mode):
     00020301  CR

Conversion queue (Lock ID / Gr/Rq mode):
     095B00F2  PW/EX

Waiting queue (Lock ID / Rq mode):
     054400BC  EX
   .
   .
   .
```

The SHOW RESOURCE command displays information taken from the RSBs of all resources in the system. For instance, the RSB at $807EB9E0_{16}$ is a parent block with no sub-RSBs.

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–9.

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  [/OWNED] [ /BRIEF  ] [ /DYNAMIC ]
                         [ /FULL   ] [ /STATIC  ]

                         [ name                 ]
                         [ /ADDRESS=expression  ]
                         [ /INDEX=expression    ]
```

Parameter

name
Name of the spin lock, fork lock, or device lock structure to be displayed. Device lock names are of the form [node$]lock, where node optionally indicates the VMScluster node name (allocation class) and lock indicates the device and controller identification (for example, HAETAR$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–18 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 *OpenVMS Alpha Device Support: Developer's Guide* for a full discussion of the role of spin locks in maintaining synchronization of kernel mode activities in a multiprocessing environment.

Table SDA–18 Static Spin Locks

Name	Description
QUEUEAST	Fork lock for queuing ASTs at IPL 6
FILSYS	Lock on file system 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
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–18 (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.

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

EMB                                 Address   80424480
Owner CPU ID           None         DIPL      0000001F
Ownership Depth    00000000         Rank      00000000
CPUs Waiting       00000000         Index     00000020

MEGA                                Address   80424500
Owner CPU ID           None         DIPL      00000016
Ownership Depth    00000000         Rank      00000002
CPUs Waiting       00000000         Index     00000022

HWCLK                               Address   80424580
Owner CPU ID           None         DIPL      00000016
Ownership Depth    00000000         Rank      00000004
CPUs Waiting       00000000         Index     00000024

   .
   .
   .
System dynamic spinlock structures
----------------------------------
OPA                                 Address   8041E880
Owner CPU ID           None         DIPL      00000014
Ownership Depth    00000000         Rank      FFFFFFFF
CPUs Waiting       00000000

MBA                                 Address   80424780
Owner CPU ID           None         DIPL      0000000B
Ownership Depth    00000000         Rank      0000000C
CPUs Waiting       00000000         Index     0000002C

NLA                                 Address   80424780
Owner CPU ID           None         DIPL      0000000B
Ownership Depth    00000000         Rank      0000000C
CPUs Waiting       00000000         Index     0000002C

PKI                                 Address   80552800
Owner CPU ID           None         DIPL      00000014
Ownership Depth    00000000         Rank      FFFFFFFF
CPUs Waiting       00000000

   .
   .
   .
```

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 /qualifier[,...]]

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.

If a stack is empty, the display shows the following:

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

Example

```
SDA> SHOW STACK

Current Operating Stack (SYSTEM):
                        FFFFFFFF8244BD08  FFFFFFFF 800600FC  SCH$REPORT_EVENT_C+000FC
                        FFFFFFFF8244BD10  00000000 00000002
                        FFFFFFFF8244BD18  00000000 00000005
                        FFFFFFFF8244BD20  FFFFFFFF 8060C7C0
                  SP => FFFFFFFF8244BD28  FFFFFFFF 8244BEE8
                        FFFFFFFF8244BD30  FFFFFFFF 80018960  EXE$HWCLKINT_C+00260
                        FFFFFFFF8244BD38  00000000 000001B8
                        FFFFFFFF8244BD40  00000000 00000050
                        FFFFFFFF8244BD48  00000000 00000210  UCB$N_RSID+00002
                        FFFFFFFF8244BD50  00000000 00000000
                        FFFFFFFF8244BD58  00000000 00000000
                        FFFFFFFF8244BD60  FFFFFFFF 804045D0  SCH$GQ_IDLE_CPUS
                        FFFFFFFF8244BD68  FFFFFFFF 8041A340  EXE$GL_FKWAITFL+00020
                        FFFFFFFF8244BD70  00000000 00000250  UCB$T_MSGDATA+00034
                        FFFFFFFF8244BD78  00000000 00000001
CHF$IS_MCH_ARGS         FFFFFFFF8244BD80  00000000 0000002B
CHF$PH_MCH_FRAME        FFFFFFFF8244BD88  FFFFFFFF 8244BFB0
CHF$IS_MCH_DEPTH        FFFFFFFF8244BD90  80000000 FFFFFFFD  G
CHF$PH_MCH_DADDR        FFFFFFFF8244BD98  00000000 00001600  CTL$C_CLIDATASZ+00060
CHF$PH_MCH_ESF_ADDR     FFFFFFFF8244BDA0  FFFFFFFF 8244BF40
CHF$PH_MCH_SIG_ADDR     FFFFFFFF8244BDA8  FFFFFFFF 8244BEE8
CHF$IH_MCH_SAVR0        FFFFFFFF8244BDB0  FFFFFFFF 8041FB00  SMP$RELEASEL+00640
CHF$IH_MCH_SAVR1        FFFFFFFF8244BDB8  00000000 00000000
CHF$IH_MCH_SAVR16       FFFFFFFF8244BDC0  00000000 0000000D
CHF$IH_MCH_SAVR17       FFFFFFFF8244BDC8  0000FFF0 00007E04
CHF$IH_MCH_SAVR18       FFFFFFFF8244BDD0  00000000 00000000
CHF$IH_MCH_SAVR19       FFFFFFFF8244BDD8  00000000 00000001
CHF$IH_MCH_SAVR20       FFFFFFFF8244BDE0  00000000 00000000
CHF$IH_MCH_SAVR21       FFFFFFFF8244BDE8  FFFFFFFF 805AE4B6  SISR+0006E
CHF$IH_MCH_SAVR22       FFFFFFFF8244BDF0  00000000 00000001
CHF$IH_MCH_SAVR23       FFFFFFFF8244BDF8  00000000 00000010
CHF$IH_MCH_SAVR24       FFFFFFFF8244BE00  00000000 00000008
CHF$IH_MCH_SAVR25       FFFFFFFF8244BE08  00000000 00000010
CHF$IH_MCH_SAVR26       FFFFFFFF8244BE10  00000000 00000001
CHF$IH_MCH_SAVR27       FFFFFFFF8244BE18  00000000 00000000
CHF$IH_MCH_SAVR28       FFFFFFFF8244BE20  FFFFFFFF 804045D0  SCH$GQ_IDLE_CPUS
                        FFFFFFFF8244BE28  30000000 00000300  UCB$L_PI_SVA
                        FFFFFFFF8244BE30  FFFFFFFF 80040F6C  EXE$REFLECT_C+00950
                        FFFFFFFF8244BE38  18000000 00000300  UCB$L_PI_SVA
                        FFFFFFFF8244BE40  FFFFFFFF 804267A0  EXE$CONTSIGNAL+00228
                        FFFFFFFF8244BE48  00000000 7FFD00A8  PIO$GW_IIOIMPA
                        FFFFFFFF8244BE50  00000003 00000000
                        FFFFFFFF8244BE58  FFFFFFFF 8003FC20  EXE$CONNECT_SERVICES_C+00920
                        FFFFFFFF8244BE60  FFFFFFFF 8041FB00  SMP$RELEASEL+00640
                        FFFFFFFF8244BE68  00000000 00000000
                        FFFFFFFF8244BE70  FFFFFFFF 8042CD50  SCH$WAIT_PROC+00060
                        FFFFFFFF8244BE78  00000000 0000000D
                        FFFFFFFF8244BE80  0000FFF0 00007E04
                        FFFFFFFF8244BE88  00000000 00000000
                        FFFFFFFF8244BE90  00000000 00000001
                        FFFFFFFF8244BE98  00000000 00000000
                        FFFFFFFF8244BEA0  FFFFFFFF 805AE4B6  SISR+0006E
                        FFFFFFFF8244BEA8  00000000 00000001
                        FFFFFFFF8244BEB0  00000000 00000010
                        FFFFFFFF8244BEB8  00000000 00000008
                        FFFFFFFF8244BEC0  00000000 00000010
                        FFFFFFFF8244BEC8  00000000 00000001
                        FFFFFFFF8244BED0  00000000 00000000
                        FFFFFFFF8244BED8  FFFFFFFF 804045D0  SCH$GQ_IDLE_CPUS
                        FFFFFFFF8244BEE0  00000000 00000001
```

```
CHF$L_SIG_ARGS          FFFFFFFF8244BEE8  0000000C 00000005
CHF$L_SIG_ARG1          FFFFFFFF8244BEF0  FFFFFFFC 00010000  SYS$K_VERSION_08
                        FFFFFFFF8244BEF8  00000300 FFFFFFFC  UCB$L_PI_SVA
                        FFFFFFFF8244BF00  00000002 00000001
                        FFFFFFFF8244BF08  00000000 0000000C
                        FFFFFFFF8244BF10  00000000 00000000
                        FFFFFFFF8244BF18  00000000 FFFFFFFC
                        FFFFFFFF8244BF20  00000008 00000000
                        FFFFFFFF8244BF28  00000000 00000001
                        FFFFFFFF8244BF30  00000008 00000000
                        FFFFFFFF8244BF38  00000000 FFFFFFFC
INTSTK$Q_R2             FFFFFFFF8244BF40  FFFFFFFF 80404668  SCH$GL_ACTIVE_PRIORITY
INTSTK$Q_R3             FFFFFFFF8244BF48  FFFFFFFF 8042F280  SCH$WAIT_KERNEL_MODE
INTSTK$Q_R4             FFFFFFFF8244BF50  FFFFFFFF 80615F00
INTSTK$Q_R5             FFFFFFFF8244BF58  00000000 00000000
INTSTK$Q_R6             FFFFFFFF8244BF60  FFFFFFFF 805AE000
INTSTK$Q_R7             FFFFFFFF8244BF68  00000000 00000000
INTSTK$Q_PC             FFFFFFFF8244BF70  00000000 FFFFFFFC
INTSTK$Q_PS             FFFFFFFF8244BF78  30000000 00000300  UCB$L_PI_SVA
                        FFFFFFFF8244BF80  FFFFFFFF 80404668  SCH$GL_ACTIVE_PRIORITY
                        FFFFFFFF8244BF88  00000000 7FFD00A8  PIO$GW_IIOIMPA
                        FFFFFFFF8244BF90  00000000 00000000
                        FFFFFFFF8244BF98  FFFFFFFF 8042CD50  SCH$WAIT_PROC+00060
                        FFFFFFFF8244BFA0  00000000 00000044
                        FFFFFFFF8244BFA8  FFFFFFFF 80403C30  SMP$GL_FLAGS
Prev SP (8244BFB0) ==>  FFFFFFFF8244BFB0  FFFFFFFF 8042CD50  SCH$WAIT_PROC+00060
                        FFFFFFFF8244BFB8  00000000 00000000
                        FFFFFFFF8244BFC0  FFFFFFFF 805EE040
                        FFFFFFFF8244BFC8  FFFFFFFF 8006DB54  PROCESS_MANAGEMENT_NPRO+0DB54
                        FFFFFFFF8244BFD0  FFFFFFFF 80404668  SCH$GL_ACTIVE_PRIORITY
                        FFFFFFFF8244BFD8  FFFFFFFF 80615F00
                        FFFFFFFF8244BFE0  FFFFFFFF 8041B220  SCH$RESOURCE_WAIT
                        FFFFFFFF8244BFE8  00000000 00000044
                        FFFFFFFF8244BFF0  FFFFFFFF 80403C30  SMP$GL_FLAGS
                        FFFFFFFF8244BFF8  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–19 for each active process in the system.

Table SDA–19 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

(continued on next page)

Table SDA–19 (Cont.) Process Information in the SHOW SUMMARY Display

Column	Contents
State	Current state of the process, which is one of the following 14 states:

State	Meaning
COM	Computable and resident in memory
COMO	Computable, but outswapped
CUR	Currently executing
CEF	Waiting for a common event flag
LEF	Waiting for a local event flag
LEFO	Outswapped and waiting for a local event flag
HIB	Hibernating
HIBO	Hibernating and outswapped
SUSP	Suspended
SUSPO	Suspended and outswapped
PFW	Waiting for a page that is not in memory (page-fault wait)
FPG	Waiting to add a page to its working set (free-page wait)
COLPG	Waiting for a page collision to be resolved (collided-page wait); this usually occurs when several processes cause page faults on the same shared page
MWAIT	Waiting for a system resource (miscellaneous wait)

Pri	Current scheduling priority of the process
PCB	Address of the process control block
PHD	Address of the process header
Wkset	Number (in decimal) of pages currently in the process working set

Example

```
SDA> SHOW SUMMARY/IMAGE
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/IMAGE command describes all active processes in the system at the time of the system failure. Note that the process NETACP is in the CUR state 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 = 80000000 :  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$L_BUGCHK_FLAGS          = FFFFFFFF804031E8 : 00000001
BUG$L_FATAL_SPSAV           = FFFFFFFF804031F0 : 00000001
BUG$REBOOT                  = FFFFFFFF8042E320 : 00001808
BUG$REBOOT_C                = FFFFFFFF8004f4D0 : 00000000

Symbols sorted by value
-----------------------
BUG$REBOOT_C                = FFFFFFFF8004f4D0 : 00000000
BUG$L_BUGCHK_FLAGS          = FFFFFFFF804031E8 : 00000001
BUG$L_FATAL_SPSAV           = FFFFFFFF804031F0 : 00000001
BUG$REBOOT                  = FFFFFFFF8042E320 : 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.

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 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 FFFFFFFF80D0E6C0/list
Entry      Address              Flink                Blink
-----      -------              ------               -----
Header     FFFFFFFF80D0E6C0     FFFFFFFF80D03780     FFFFFFFF80D0E800
    1.     FFFFFFFF80D0E790     FFFFFFFF80D0E7C0     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.
```

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    DEC 7000 Model 610      Primary CPU ID 00
Cycle Time     5.5 nsec (181 MHz)      Pagesize       8192 Byte

Memory Configuration:
Cluster    PFN Start    PFN Count         Range (MByte)       Usage
 #01             0          256         0.0 MB -    2.0 MB    Console
 #02           256        16128         2.0 MB -  128.0 MB    System

Per-CPU Slot Processor Information:
CPU ID         00                      CPU State    rc,pa,pp,cv,pv,pmv,pl
CPU Type       EV4  P3.0               Halt PC      00000000 20000000
PAL Code       5.41                    Halt PS      00000000 00001F00
CPU Revision   ....                    Halt Code    00000000 00000000
Serial Number  GA30366899                           Bootstrap or Powerfail

Adapter Configuration:
----------------------
TR  Adapter Name (Address)  Hose  Bus    Node  Device Name     HW-Id/SW
--  ----------------------  ----  ------ ----  --------------  --------
 1  KA0302      (805C9FC0)     0  LSB
                                          0  KA0302_EV4_4MB    00008001
                                          7  KA0302_MEM        00004000
                                          8  KA0302_IOP        00002000
 2  XMI         (805CA380)     0  XMI
                                          1  DEMNA             08020C03
                                          2  KDM70             BB110C22
                                          4  XZA_SCSI          413F0C36
                                          5  XZA_SCSI          413F0C36
                                          8  LAMB              0105102A
                                         13  CIMNA             01110C2F
                                         14  DEMFA             05130823
 3  DEMNA       (805CA840)     0  GENXMI
                                          0  DEMNA             00000C03
 4  KDM70       (805CAA80)     0  KDM70
                                          0  KDM70             00000C22
 5  XZA         (805CB600)     0  SCSI
```

```
Adapter Configuration:
----------------------
TR  Adapter Name (Address)  Hose  Bus    Node  Device Name     HW-Id/SW
--  ----------------------  ----  -----  ----  --------------  --------
                                            0  XZA_SCSI          00000C36
                                            1  XZA_SCSI          00000C36
 6  XZA         (805CBA40)     0  SCSI
                                            0  XZA_SCSI          00000C36
                                            1  XZA_SCSI          00000C36
 7  CIMNA       (805CBEC0)     0  CI
                                            0  CIMNA             00000C2F
 8  GENXMI      (805CC200)     0  GENXMI
```

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
Crashdump Summary Information:
------------------------------
Crash Time:        11-MAY-1994 00:44:38.97
Bugcheck Type:     UNXSIGNAL, Unexpected signal name in ACP
Node:              CLAWS   (Clustered)
CPU Type:          DEC 7000 Model 610
VMS Version:       V6.1
Current Process:   OPERATOR
Current Image:     $65$DUA3:[SYS2.SYSCOMMON.][SYSEXE]BACKUP.EXE
Failing PC:        FFFFFFFF 8484F93C
Failing PS:        38000000 00000000
Module:            F11BXQP
Offset:            0000D93C

Boot Time:         11-MAY-1994 09:03:22.00
System Uptime:               0 15:41:16.97
Crash/Primary CPU: 00/00
Saved Processes:   19
Pagesize:          8 KByte (8192 bytes)
Physical Memory:   128 MByte (16384 PFNs)
Dumpfile Pagelets: 83405 blocks

Dump Flags:        olddump,writecomp,errlogcomp,dump_style
EXE$GL_FLAGS:      poolpging,init,bugdump
```

```
Stack Pointers:
KSP = 00000000 7FF4EDF0   ESP = 00000000 7FF9A000   SSP = 00000000 7FFA0100
USP = 00000000 7FE530C8

General Registers:
R0  = 00000000 0000041C   R1  = 00000000 7FF4F218   R2  = FFFFFFFF 84880590
R3  = 00000000 00000000   R4  = 00000000 7FF4F3AC   R5  = FFFFFFFF 8077FA40
R6  = 00000000 00000072   R7  = 00000000 7FF4F3BC   R8  = 00000000 7FF4F3B8
R9  = 00000000 7FF4F348   R10 = 00000000 7FF4F3AC   R11 = 00000000 7FF4F344
R12 = 00000000 7FF4F340   R13 = 00000000 00000003   R14 = FFFFFFFF 806D7780
R15 = 00000000 7FF4F610   R16 = 00000000 0000041C   R17 = 00000000 7FF4EEC0
R18 = 00000000 7FF4F080   R19 = 00000000 7FF4F028   R20 = FFFFFFFF 8487E380
R21 = FFFFFFFF 8059BF00   R22 = 00000000 00000040   R23 = FFFFFFFF 84880590
R24 = FFFFFFFF 80417C60   AI  = 00000000 7FF4F028   RA  = 00000000 7FF4EEC0
PV  = FFFFFFFF 84880590   R28 = FFFFFFFF 8485F57C   FP  = 00000000 7FF4EDF0
PC  = FFFFFFFF 8485F5E0   PS  = 30000000 00000000

Exception Frame:
R2  = FFFFFFFF 84882718   R3  = 00000000 00000000   R4  = 00000000 7FF4F3AC
R5  = FFFFFFFF 8077FA40   R6  = 00000000 00000072   R7  = 00000000 7FF4F3BC
PC  = FFFFFFFF 8484F93C   PS  = 38000000 00000000

Signal Array:
Arg Count    = 00000005
Condition    = 0000000C
Argument #2  = 00000004
Argument #3  = 00970D2B
Argument #4  = 8484F93C
Argument #5  = 00000000

Mechanism Array:
Arguments    = 0000002B                 Establisher FP = 00000000 7FF4F218
Flags        = 00000000                 Exception FP   = 00000000 7FF4F080
Depth        = 00000002                 Signal Array   = 00000000 7FF4F028
R0  = FFFFFFFF 80594F80   R1  = FFFFFFFF 80594F80   R16 = FFFFFFFF 806D7780
R17 = 00000000 7FF4F344   R18 = FFFFFFFF 8076344C   R19 = 00000000 7FF4F3AC
R20 = FFFFFFFF 8487E380   R21 = FFFFFFFF 8059BF00   R22 = 00000000 00970D27
R23 = FFFFFFFF 80CEB227   R24 = FFFFFFFF 80594F80   R25 = 00000000 00000003
R26 = FFFFFFFF 8487099C   R27 = FFFFFFFF 8487E380   R28 = 00000000 00000000

System Registers:
Page Table Base Register (PTBR)                          00000000 000005A9
Processor Base Register (PRBR)                           FFFFFFFF 80590000
Privileged Context Block Base (PCBB)                     00000000 03ADE080
System Control Block Base (SCBB)                         00000000 00000560
Software Interrupt Summary Register (SISR)               00000000 00000000
Address Space Number (ASN)                               00000000 0000003F
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)                  00000002 00000000

Failing Instruction:
F11BXQP_PRO+0193C:   STL  R24,#X0004(R22)
```

```
Instruction Stream (last 20 instructions):
F11BXQP_PRO+018EC:   LDA  SP,#XFFE0(SP)
F11BXQP_PRO+018F0:   BIS  R31,R17,R0
F11BXQP_PRO+018F4:   STQ  R27,(SP)
F11BXQP_PRO+018F8:   BIS  R31,R10,R19
F11BXQP_PRO+018FC:   STQ  R26,#X0010(SP)
F11BXQP_PRO+01900:   BIS  R31,R27,R20
F11BXQP_PRO+01904:   STQ  FP,#X0018(SP)
F11BXQP_PRO+01908:   BIS  R31,SP,FP
F11BXQP_PRO+0190C:   BIS  R31,R0,R1
F11BXQP_PRO+01910:   LDA  R17,#XFF98(R19)
F11BXQP_PRO+01914:   LDL  R21,#X0014(R16)
F11BXQP_PRO+01918:   BIS  R31,R1,R24
F11BXQP_PRO+0191C:   LDL  R22,(R21)
F11BXQP_PRO+01920:   XOR  R21,R22,R23
F11BXQP_PRO+01924:   BNE  R23,#X000005
F11BXQP_PRO+01928:   STL  R24,#X0004(R21)
F11BXQP_PRO+0192C:   STL  R24,(R21)
F11BXQP_PRO+01930:   STL  R21,#X0004(R24)
F11BXQP_PRO+01934:   STL  R21,(R24)
F11BXQP_PRO+01938:   BR  R31,#X000004
F11BXQP_PRO+0193C:   STL  R24,#X0004(R22)
```

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.

Qualifiers

/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

Displays machine-check information, including PALcode-specific, processor-specific, and system-specific information.

Format

CLUE MCHK

Parameters

None.

Qualifiers

None.

Description

The CLUE MCHK command displays machine-check information, including PALcode-specific, processor-specific, and system-specific information.

Example

```
SDA> CLUE MCHK

Machine Check Information:
--------------------------
Mchk Frame     83760000     Frame Size     000001E8     Frame Flags     00000000
Mchk Ident     00000088     CPU Offset     00000118     System Offset   000001A8

PALcode Temporary Registers:
R0  = 00000000 00000001     R1  = 00000000 00000000     R2  =  000044F8 00000004
R3  = 00000000 00ED25B0     R4  = 00000000 00000001     R5  =  00000000 00426380
R6  = 00000000 00314FD0     R7  = 00000000 000F4000     R8  =  00000000 00000000
R9  = 00000000 0000001B     R10 = 0000459E 253B8997     R11 =  00000000 00000000
R12 = 00000000 00000000     R13 = FFFFFFFF 8042F548     R14 =  00000000 00315900
R15 = 00000000 00000018     R16 = 00000000 00000001     R17 =  00000000 00000000
R18 = 00000000 00000000     R19 = 00000000 00000000     R20 =  00000000 0000001B
R21 = 00000000 004DC4A0     R22 = FFFFFFFF FFF929C4     R23 =  00000000 00000001
R24 = FFFFFFFF 80570000     R25 = 00000000 00010000     R26 =  00000000 7FF96000
R27 = 00000000 00000000     R28 = 00000000 0143A000     R29 =  00000002 00000000
R30 = 00000000 00900000     R31 = 00000000 01556080

Processor Specific Information:
Exception Address     00000000 0038219A    Exception Summary    00000000 00000000
PAL Base Address      00000000 00054000    Exception Mask       00000000 00000000
HW Interrupt Request  00000000 00000300    HW Interrupt Enable  00000001 FFFFD8F0
MM_CSR                00000000 00005000    ICCSR                00000000 00000000
D-Cache Address       00000007 FFFFFFFF    D-Cache Status       00000000 00004238
BIU Status            00000000 00007340    BIU Address [7..0]   00000000 014054C8
BIU Control           0000000E 20006447    Fill Address         00000000 014054C0
Single-Bit Syndrome   00000000 00000052    Processor Mchk VA    00000000 000F8190
A-Box Control         00000000 0000040E    B-Cache TAG          00000000 00001415

System Specific Information:
Exception Ident       00000000 00000088    Mem Conf Register    11111111 11808080
IO Slot Conf Reg      00000000 00140000    Failing Address      00000000 02000018
TC Config Register    00000000 00000016    TC Error Register    00000000 0F000200
Interrupt Register    00000000 0007FE00    Interrupt Mask Reg   00000000 00000010
```

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):
---------------------------
                                                  # Process
Index  Type  Total Size       Free Reservable  Page Swap  Flags
-----  ----  ---------- ---------- ----------  ---- ----  ----------
  1    Swap       38272      38272      38272     0    0  inited
  3    Page      270080     270080     190912    20    0  inited
```

This example shows the display produced by the CLUE MEMORY/FILES command.

2. SDA> CLUE MEMORY/FREE/FULL

```
Nonpaged Dynamic Pool - Free Packet Queue:
------------------------------------------

CLASSDR  80B1A380 :    64646464 64646464 00000040 80B1A480  .¤±.@...dddddddd
CLASSDR  80B1A480 :    64646464 64646464 00000040 80B43C80  .<´.@...dddddddd
IPC      80B43C80 :    8016B950 207B0040 00000040 80B50640  @.µ.@...@.{ P¹..
ACB      80B50640 :    8008AF6C 03020024 00000040 80B52E80  ..µ.@...$...l¯..
ACB      80B52E80 :    8008AF6C 03020024 00000040 80B5FC00  .üµ.@...$...l¯..
ACB      80B5FC00 :    8008AF6C 03020024 00000040 80B610C0  À.¶.@...$...l¯..
ACB      80B610C0 :    8008AF6C 03020024 00000040 80B66CC0  Àl¶.@...$...l¯..
ACB      80B66CC0 :    8008AF6C 03020024 00000040 80B704C0  À.·.@...$...l¯..
CLASSDR  80B704C0 :    64646464 64646464 00000080 80B72EC0  À.·.....dddddddd
CLASSDR  80B72EC0 :    64646464 64646464 00000080 80B73700  .7·.....dddddddd
CLASSDR  80B73700 :    64646464 64646464 00000080 80B752C0  ÀR·.....dddddddd
TWP      80B752C0 :    8005C30C 3A300080 00000080 80B77A80  .z·.......0:.Ã..
CLASSDR  80B77A80 :    64646464 64646464 00000040 80B79380  ..·.@...ddddddddd
CLASSDR  80B79380 :    64646464 64646464 00000040 80B79C80  ..·.@...ddddddddd
CLASSDR  80B79C80 :    64646464 64646464 00000080 80B7AB40  @«·.....dddddddd
CLASSDR  80B7AB40 :    64646464 64646464 00000040 80B7B640  @¶·.@...ddddddddd
CLASSDR  80B7B640 :    64646464 64646464 00000080 80B7C180  .Á·.....dddddddd
CLASSDR  80B7C180 :    64646464 64646464 00000040 80B7C280  .Â·.@...ddddddddd
CDRP     80B7C280 :    8014C730 3A390088 000000C0 80B7CC00  .Ì·.À.....9:0Ç..
CLASSDR  80B7CC00 :    64646464 64646464 000000C0 80B81100  ..¸.À...ddddddddd
CLASSDR  80B81100 :    64646464 64646464 00000080 80B81A00  ..¸.....dddddddd
CLASSDR  80B81A00 :    64646464 64646464 000000C0 80B81B80  ..¸.À...ddddddddd
CLASSDR  80B81B80 :    64646464 64646464 00000040 80B82740  @'¸.@...ddddddddd
CLASSDR  80B82740 :    64646464 64646464 00000100 80B83680  .6¸.....dddddddd
ACB      80B83680 :    8008AF6C 03020024 00000040 80B84340  @C¸.@...$...l¯..
CLASSDR  80B84340 :    64646464 64646464 00000040 80B85380  .S¸.@...ddddddddd
CIMSG    80B85380 :    8025660C 003C00C0 000000C0 80B86580  .e¸.À...À.<..f%.
CLASSDR  80B86580 :    64646464 64646464 00000240 80B87640  @v¸.@...ddddddddd
CXB      80B87640 :    802B686C 611B02C0 00000440 80B885C0  À.¸.@...À..alh+.
DSRV     80B885C0 :    8019B6F4 036900C0 000000C0 80B88CC0  À.¸.À...À.i.ô¶..
CIMSG    80B88CC0 :    8025660C 003C00C0 00000240 80B89000  ..¸.@...À.<..f%.
CLASSDR  80B89000 :    64646464 64646464 00000040 80B89900  ..¸.@...ddddddddd
CXB      80B89900 :    8005CD20 001B0740 00000740 80B8A780  .§¸.@...@... Í..
CXB      80B8A780 :    8005CEC0 001B0740 00000E80 80B8BC40  @¼¸.....@...ÀÎ..
CXB      80B8BC40 :    802B686C 611B02C0 000002C0 80B8C1C0  ÀÁ¸.À...À..alh+.
CLASSDR  80B8C1C0 :    64646464 64646464 00003E40 00000000  ....@>..ddddddddd
Free Packet Queue, Status: Valid, 36 elements
```

The CLUE MEMORY/FULL/FREE 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     FFFFFFFF80000000  FFFFFFFF80346000  Current Size   419/ 419
    Base/End PA     FFFFFFFF00400000  FFFFFFFF00746000  Free             /   0
    Total Size      FFFFFFFF00346000            3.2 MB  In Use           / 419
    Bitmap VA/Size  FFFFFFFF80A12460  FFFFFFFF00000040  Initial Size   512/ 512
    Slice Size      FFFFFFFF00002000                    Released        93/  93
    Next free Slice FFFFFFFF000001A3

    Image                          Base             End              Length
    SYS$PUBLIC_VECTORS        FFFFFFFF80000000  FFFFFFFF80001600    00001600
    SYS$BASE_IMAGE            FFFFFFFF80002000  FFFFFFFF8000CA00    0000AA00
    SYS$PNBTDRIVER            FFFFFFFF8000E000  FFFFFFFF80016800    00008800
    SYS$OPDRIVER              FFFFFFFF80018000  FFFFFFFF8001BC00    00003C00
    SYSTEM_PRIMITIVES         FFFFFFFF8001C000  FFFFFFFF80036600    0001A600
    SYSTEM_SYNCHRONIZATION    FFFFFFFF80038000  FFFFFFFF80041A00    00009A00
    ERRORLOG                  FFFFFFFF80042000  FFFFFFFF80044E00    00002E00
    SYS$CPU_ROUTINES_0302     FFFFFFFF80046000  FFFFFFFF8005E200    00018200
    EXCEPTION                 FFFFFFFF80060000  FFFFFFFF8006C400    0000C400
    IO_ROUTINES               FFFFFFFF8006E000  FFFFFFFF80089E00    0001BE00
    SYSDEVICE                 FFFFFFFF8008A000  FFFFFFFF8008DC00    00003C00
    PROCESS_MANAGEMENT        FFFFFFFF8008E000  FFFFFFFF800A3400    00015400
    SYS$VM                    FFFFFFFF800A4000  FFFFFFFF800C3E00    0001FE00
    SHELL8K                   FFFFFFFF800C4000  FFFFFFFF800C4E00    00000E00
    LOCKING                   FFFFFFFF800C6000  FFFFFFFF800D7000    00011000
    MESSAGE_ROUTINES          FFFFFFFF800D8000  FFFFFFFF800DE600    00006600
    LOGICAL_NAMES             FFFFFFFF800E0000  FFFFFFFF800E2800    00002800
    F11BXQP                   FFFFFFFF800E4000  FFFFFFFF800EA800    00006800
    IMAGE_MANAGEMENT          FFFFFFFF800EC000  FFFFFFFF800EEC00    00002C00
    SECURITY                  FFFFFFFF800F0000  FFFFFFFF800F9600    00009600
    SYSGETSYI                 FFFFFFFF800FA000  FFFFFFFF800FB200    00001200
    SYS$TRANSACTION_SERVICES  FFFFFFFF800FC000  FFFFFFFF8011E600    00022600
    SYS$UTC_SERVICES          FFFFFFFF80120000  FFFFFFFF80121200    00001200
    SYS$VCC                   FFFFFFFF80122000  FFFFFFFF8012DA00    0000BA00
    SYS$SCS                   FFFFFFFF8012E000  FFFFFFFF80139000    0000B000
    SYS$CLUSTER               FFFFFFFF8013A000  FFFFFFFF8016C600    00032600
    SYS$IPC_SERVICES          FFFFFFFF8016E000  FFFFFFFF801AAC00    0003CC00
    MSCP                      FFFFFFFF801AC000  FFFFFFFF801B2E00    00006E00
    SYSLDR_DYN                FFFFFFFF801B4000  FFFFFFFF801B5400    00001400
    SYS$TTDRIVER              FFFFFFFF801B6000  FFFFFFFF801C8000    00012000
    SYS$PNDRIVER              FFFFFFFF801C8000  FFFFFFFF801E1600    00019600
    SYS$DUDRIVER              FFFFFFFF801E2000  FFFFFFFF801F0C00    0000EC00
    SYS$SHDRIVER              FFFFFFFF801F2000  FFFFFFFF80233A00    00041A00
    RMS                       FFFFFFFF80234000  FFFFFFFF802A7800    00073800
    SYS$PUDRIVER              FFFFFFFF802A8000  FFFFFFFF802AE200    00006200
    SYS$EXDRIVER              FFFFFFFF802B0000  FFFFFFFF802C6800    00016800
    SYS$PEDRIVER              FFFFFFFF802C8000  FFFFFFFF802E7200    0001F200
    SYS$PKZDRIVER             FFFFFFFF802E8000  FFFFFFFF802F3600    0000B600
    SYS$DKDRIVER              FFFFFFFF802F4000  FFFFFFFF802FC000    00008000
    SYS$TUDRIVER              FFFFFFFF802FC000  FFFFFFFF8030D600    00011600
    SYS$FTDRIVER              FFFFFFFF8030E000  FFFFFFFF80310000    00002000
    SYS$MKDRIVER              FFFFFFFF80310000  FFFFFFFF80315000    00005000
    NETDRIVER                 FFFFFFFF80316000  FFFFFFFF80316200    00000200
    NETDRIVER                 FFFFFFFF80318000  FFFFFFFF80330200    00018200
    NDDRIVER                  FFFFFFFF80332000  FFFFFFFF80335600    00003600
    SYS$CTDRIVER              FFFFFFFF80336000  FFFFFFFF80340E00    0000AE00
    SYS$RTTDRIVER             FFFFFFFF80342000  FFFFFFFF80345800    00003800
```

```
Execlet Data Region                                                    Pages/Slices
    Base/End VA     FFFFFFFF80800000  FFFFFFFF808B0000  Current Size    88/1408
    Base/End PA     FFFFFFFF00800000  FFFFFFFF008B0000  Free              /  28
    Total Size      FFFFFFFF000B0000    0.6 MB          In Use            /1380
    Bitmap VA/Size  FFFFFFFF80A124A0  FFFFFFFF00000100  Initial Size   128/2048
    Slice Size      FFFFFFFF00000200                    Released        40/ 640
    Next free Slice FFFFFFFF00000564

    Image                          Base              End               Length
    SYS$PUBLIC_VECTORS          FFFFFFFF80800000  FFFFFFFF80804200  00004200
    SYS$BASE_IMAGE              FFFFFFFF80804200  FFFFFFFF8081F000  0001AE00
    SYS$PNBTDRIVER              FFFFFFFF8081F000  FFFFFFFF80822600  00003600
    SYS$OPDRIVER                FFFFFFFF80822600  FFFFFFFF80823000  00000A00
    SYSTEM_PRIMITIVES           FFFFFFFF80823000  FFFFFFFF80829800  00006800
    SYSTEM_SYNCHRONIZATION      FFFFFFFF80829800  FFFFFFFF8082B600  00001E00
    ERRORLOG                    FFFFFFFF8082B600  FFFFFFFF8082BC00  00000600
    SYS$CPU_ROUTINES_0302       FFFFFFFF8082BC00  FFFFFFFF80833400  00007800
    EXCEPTION                   FFFFFFFF80833400  FFFFFFFF80838800  00005400
    IO_ROUTINES                 FFFFFFFF80838800  FFFFFFFF8083E000  00005800
    SYSDEVICE                   FFFFFFFF8083E000  FFFFFFFF8083EC00  00000C00
    PROCESS_MANAGEMENT          FFFFFFFF8083EC00  FFFFFFFF80843C00  00005000
    SYS$VM                      FFFFFFFF80843C00  FFFFFFFF80848200  00004600
    SHELL8K                     FFFFFFFF80848200  FFFFFFFF80849000  00000E00
    LOCKING                     FFFFFFFF80849000  FFFFFFFF8084AC00  00001C00
    MESSAGE_ROUTINES            FFFFFFFF8084AC00  FFFFFFFF8084C400  00001800
    LOGICAL_NAMES               FFFFFFFF8084C400  FFFFFFFF8084D800  00001400
    F11BXQP                     FFFFFFFF8084D800  FFFFFFFF8084EA00  00001200
    SYSLICENSE                  FFFFFFFF8084EA00  FFFFFFFF8084EE00  00000400
    IMAGE_MANAGEMENT            FFFFFFFF8084EE00  FFFFFFFF8084F600  00000800
    SECURITY                    FFFFFFFF8084F600  FFFFFFFF80852200  00002C00
    SYSGETSYI                   FFFFFFFF80852200  FFFFFFFF80852400  00000200
    SYS$TRANSACTION_SERVICES    FFFFFFFF80852400  FFFFFFFF80858E00  00006A00
    SYS$UTC_SERVICES            FFFFFFFF80858E00  FFFFFFFF80859400  00000600
    SYS$VCC                     FFFFFFFF80859400  FFFFFFFF8085AE00  00001A00
    SYS$SCS                     FFFFFFFF8085AE00  FFFFFFFF8085C600  00001800
    SYS$CLUSTER                 FFFFFFFF8085C600  FFFFFFFF80864A00  00008400
    SYS$IPC_SERVICES            FFFFFFFF80864A00  FFFFFFFF8086A000  00005600
    MSCP                        FFFFFFFF8086A000  FFFFFFFF8086B600  00001600
    SYSLDR_DYN                  FFFFFFFF8086B600  FFFFFFFF8086CC00  00001600
    SYS$TTDRIVER                FFFFFFFF8086CC00  FFFFFFFF8086F400  00002800
    SYS$PNDRIVER                FFFFFFFF8086F400  FFFFFFFF80874600  00005200
    SYS$DUDRIVER                FFFFFFFF80874600  FFFFFFFF80877400  00002E00
    SYS$SHDRIVER                FFFFFFFF80877400  FFFFFFFF80877600  00000200
    SYS$SHDRIVER                FFFFFFFF80877600  FFFFFFFF80880E00  00009800
    RMS                         FFFFFFFF80880E00  FFFFFFFF80894200  00013400
    RECOVERY_UNIT_SERVICES      FFFFFFFF80894200  FFFFFFFF80894600  00000400
    SYS$PUDRIVER                FFFFFFFF80894600  FFFFFFFF80895A00  00001400
    SYS$EXDRIVER                FFFFFFFF80895A00  FFFFFFFF80897E00  00002400
    SYS$PEDRIVER                FFFFFFFF80897E00  FFFFFFFF8089E600  00006800
    SYS$PKZDRIVER               FFFFFFFF8089E600  FFFFFFFF808A0200  00001C00
    SYS$DKDRIVER                FFFFFFFF808A0200  FFFFFFFF808A1E00  00001C00
    SYS$TUDRIVER                FFFFFFFF808A1E00  FFFFFFFF808A2000  00000200
    SYS$TUDRIVER                FFFFFFFF808A2000  FFFFFFFF808A5200  00003200
    SYS$FTDRIVER                FFFFFFFF808A5200  FFFFFFFF808A5A00  00000800
    SYS$MKDRIVER                FFFFFFFF808A5A00  FFFFFFFF808A6C00  00001200
    NETDRIVER                   FFFFFFFF808A6C00  FFFFFFFF808A9600  00002A00
    NDDRIVER                    FFFFFFFF808A9600  FFFFFFFF808AA000  00000A00
    SYS$CTDRIVER                FFFFFFFF808AA000  FFFFFFFF808ABE00  00001E00
    SYS$RTTDRIVER               FFFFFFFF808ABE00  FFFFFFFF808AC800  00000A00
    28 free Slices              FFFFFFFF808AC800  FFFFFFFF808B0000  00003800
```

```
VMS Exec Data Region                                                    Pages/Slices
    Base/End VA     FFFFFFFF80900000  FFFFFFFF80B50000  Current Size    296/296
    Base/End PA     FFFFFFFF00900000  FFFFFFFF00B50000  Free              /  1
    Total Size      FFFFFFFF00250000            2.3 MB  In Use            /295
    Bitmap VA/Size  FFFFFFFF80A125A0  FFFFFFFF00000028  Initial Size     96/296
    Slice Size      FFFFFFFF00002000                    Released          0/  0
    Next free Slice FFFFFFFF00000085

    Item                              Base             End          Length
    System Header                 FFFFFFFF80900000  FFFFFFFF80904000  00004000
    PFN Database                  FFFFFFFF80904000  FFFFFFFF80A04000  00100000
    Error Log Allocation Buffers  FFFFFFFF80A04000  FFFFFFFF80A06000  00002000
    3 free Slices                 FFFFFFFF80A06000  FFFFFFFF80A0C000  00006000
    Nonpaged Pool (initial size)  FFFFFFFF80A0C000  FFFFFFFF80B50000  00144000

Resident Image Code Region                                              Pages/Slices
    Base/End VA     FFFFFFFF80400000  FFFFFFFF805CE000  Current Size    231/231
    Base/End PA     FFFFFFFF00C00000  FFFFFFFF00DCE000  Free              /  0
    Total Size      FFFFFFFF001CE000    1.8 MB          In Use            /231
    Bitmap VA/Size  FFFFFFFF80A125C8  FFFFFFFF00000040  Initial Size    512/512
    Slice Size      FFFFFFFF00002000                    Released        281/281
    Next free Slice FFFFFFFF000000E7

    Image                                   Base             End           Length
    DPML$SHR                            FFFFFFFF80400000  FFFFFFFF804B6600  000B6600
    DECC$SHR                            FFFFFFFF804B8000  FFFFFFFF80541600  00089600
    DECC$SHR                            FFFFFFFF80542000  FFFFFFFF80542400  00000400
    LIBRTL                              FFFFFFFF80544000  FFFFFFFF805BEE00  0007AE00
    LIBOTS                              FFFFFFFF805C0000  FFFFFFFF805CDC00  0000DC00
```

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
Code Huge Page                            FFFFFFFF80000000  FFFFFFFF80400000  00400000
Data Huge Page                            FFFFFFFF80400000  FFFFFFFF80500000  00100000
System Header                             FFFFFFFF80500000  FFFFFFFF80506000  00006000
PFN Database                              FFFFFFFF80506000  FFFFFFFF80586000  00080000
Error Log Allocation Buffers              FFFFFFFF80586000  FFFFFFFF80588000  00002000
Nonpaged Pool (initial size)              FFFFFFFF80590000  FFFFFFFF806A0000  00110000
Nonpaged Pool Expansion Area              FFFFFFFF806A0000  FFFFFFFF80B94000  004F4000
Balance Slots                             FFFFFFFF80B94000  FFFFFFFF84374E50  037E0E50
Global Page Table (GPT)                   FFFFFFFF84374E50  FFFFFFFF8437C000  000071B0
Paged Pool                                FFFFFFFF8437C000  FFFFFFFF845FA000  0027E000
System Control Block (SCB)                FFFFFFFF845FA000  FFFFFFFF84644000  0004A000
Hardware Restart Parameter Block (HWRPB)  FFFFFFFF84644000  FFFFFFFF846458F8  000018F8
Guard Page                                FFFFFFFF84658000  FFFFFFFF8465A000  00002000
Prim CPU System Context Kernel Stack      FFFFFFFF8465A000  FFFFFFFF8465C000  00002000
Guard Page                                FFFFFFFF8465C000  FFFFFFFF8465E000  00002000
Prim CPU Machine Check Logout Area        FFFFFFFF8465E000  FFFFFFFF8465E300  00000300
Guard Page                                FFFFFFFF84660000  FFFFFFFF84662000  00002000
Lock ID Table                             FFFFFFFF8469C000  FFFFFFFF846DE000  00042000
Dumpfile Write Memory Mapping             FFFFFFFF847BA000  FFFFFFFF847CA000  00010000
Swapper Process Kernel Stack              FFFFFFFF847EE000  FFFFFFFF847F0000  00002000
Idle Loop's Mapping of Zero Pages         FFFFFFFF847F0000  FFFFFFFF847F2000  00002000
Posix Cloning Parent's Page Mapping       FFFFFFFF847FE000  FFFFFFFF84800000  00002000
Posix Cloning Child's Page Mapping        FFFFFFFF84800000  FFFFFFFF84802000  00002000
Swapper L2PT and L3PT                     FFFFFFFF8480A000  FFFFFFFF84816000  0000C000
Process Creation L1PT                     FFFFFFFF84816000  FFFFFFFF84818000  00002000
Tape Mount Verification Buffer            FFFFFFFF8493A000  FFFFFFFF8493E000  00004000
Mount Verification Buffer                 FFFFFFFF8493E000  FFFFFFFF84940000  00002000
Demand Zero Optimization Page             FFFFFFFF84940000  FFFFFFFF84942000  00002000
```

```
Erase Pattern Buffer Page                FFFFFFFF84942000  FFFFFFFF84944000  00002000
Erase Pattern Page Table Page            FFFFFFFF84944000  FFFFFFFF84946000  00002000
Executive Mode Data Page                 FFFFFFFF84946000  FFFFFFFF84948000  00002000
System Space Expansion Region            FFFFFFFF88000000  FFFFFFFFFFFE00000  77E00000
System Page Table (SPT)                  FFFFFFFFFFFE00000  FFFFFFFFFFFFFFFF  00200000
```

The CLUE MEMORY/LAYOUT command decodes and displays the system virtual address space layout.

5. SDA> CLUE MEMORY/LOOKASIDE

```
Lookaside List Queue Information:
---------------------------------
Listhead Addr: 8041FC00   Size:   64   Status: Valid, 56 elements
Listhead Addr: 8041FC08   Size:  128   Status: Valid, 6 elements
Listhead Addr: 8041FC10   Size:  192   Status: Valid, 183 elements
Listhead Addr: 8041FC18   Size:  256   Status: Valid, 138 elements
Listhead Addr: 8041FC20   Size:  320   Status: Valid, 1 element
Listhead Addr: 8041FC28   Size:  384   Status: Valid, 1 element
Listhead Addr: 8041FC30   Size:  448   Status: Valid, 1 element
Listhead Addr: 8041FC38   Size:  512   Status: Valid, 1 element
Listhead Addr: 8041FC40   Size:  576   Status: Valid, 1 element
Listhead Addr: 8041FC48   Size:  640   Status: Valid, 1 element
Listhead Addr: 8041FC50   Size:  704   Status: Valid, empty
Listhead Addr: 8041FC58   Size:  768   Status: Valid, 1 element
Listhead Addr: 8041FC60   Size:  832   Status: Valid, empty
Listhead Addr: 8041FC68   Size:  896   Status: Valid, 1 element
Listhead Addr: 8041FC70   Size:  960   Status: Valid, 1 element
Listhead Addr: 8041FC78   Size: 1024   Status: Valid, 1 element
Listhead Addr: 8041FC80   Size: 1088   Status: Valid, 1 element
Listhead Addr: 8041FC88   Size: 1152   Status: Valid, empty
Listhead Addr: 8041FC90   Size: 1216   Status: Valid, empty
Listhead Addr: 8041FC98   Size: 1280   Status: Valid, 1 element
Listhead Addr: 8041FCA0   Size: 1344   Status: Valid, 1 element
Listhead Addr: 8041FCA8   Size: 1408   Status: Valid, empty
Listhead Addr: 8041FCB0   Size: 1472   Status: Valid, 1 element
Listhead Addr: 8041FCB8   Size: 1536   Status: Valid, 1 element
Listhead Addr: 8041FCC0   Size: 1600   Status: Valid, 1 element
Listhead Addr: 8041FCC8   Size: 1664   Status: Valid, 1 element
Listhead Addr: 8041FCD0   Size: 1728   Status: Valid, empty
Listhead Addr: 8041FCD8   Size: 1792   Status: Invalid, memory access error
   Error in queue linkage at address A404FFFC, after tracing 0 elements
   Not in physical memory
Listhead Addr: 8041FCE0   Size: 1856   Status: Valid, empty
Listhead Addr: 8041FCE8   Size: 1920   Status: Valid, 1 element
Listhead Addr: 8041FCF0   Size: 1984   Status: Valid, empty
Listhead Addr: 8041FCF8   Size: 2048   Status: Valid, 1 element
Listhead Addr: 8041FD00   Size: 2112   Status: Valid, empty
[...]
```

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             39454      Successful Exp Attempts           0
    Total Page Reads              21885      Unsuccessful Exp Attempts         0
    I/O's to read Pages            9150      Expansion Failures                0
    Modified Pages Written            0      Failed Pages Accumulator          0
    I/O's to write Mod Pages          0      Total Alloc Requests          13545
    Demand Zero Faults             8296      Failed Alloc Requests             0
    Global Valid Faults            6680      Paged Pool:
    Modified Faults               15261      Total Failures                    0
    Read Faults                       0      Failed Pages Accumulator          0
    Execute Faults                 2062      Total Alloc Requests           2063
                                             Failed Alloc Requests             0

    Direct I/O                    25810      Cur Mapped Gbl Sections         246
    Buffered I/O                  34491      Max Mapped Gbl Sections         246
    Split I/O                      1664      Cur Mapped Gbl Pages           3052
    Hits                          27847      Max Mapped Gbl Pages           3058
    Logical Name Transl          162354      Maximum Processes                21
    Dead Page Table Scans             0      Sched Zero Pages Created       8562
    Memory Management Statistics:
    -----------------------------
    Distributed Lock Manager:           Local         Incoming         Outgoing
    $ENQ New Lock Requests              18635             4353            10334
    $ENQ Conversion Requests            21209               13              653
    $DEQ Dequeue Requests               17953             4165            10070
    Blocking ASTs                          17               14                1
    Directory Functions                                 541406            12580
    Deadlock Messages                                      189              120

    $ENQ Requests that Wait         349      Deadlock Searches Performed      39
    $ENQ Requests not Queued        166      Deadlocks Found                  39

    MSCP Statistics:                         Total IOs                   1947594
    Count of VC Failures              0      Split IOs                         0
    Count of Hosts Served             2      IOs that had to Wait (Buf)        0
    Count of Disks Served             9      Requests in MemWait Queue         0
    MSCP_BUFFER (SYSGEN)            128      Max Req ever in MemWait           0
    MSCP_CREDITS (SYSGEN)             8
    Memory Management Statistics:
    -----------------------------
    File System Cache:     Current SYSGEN Param        Hits      Misses Hitrate
    File Header Cache     (ACP_HDRCACHE  = 196)        3543        1090   76.4%
    Storage Bitmap Cache  (ACP_MAPCACHE  =  49)          20         529    3.6%
    Directory Data Cache  (ACP_DIRCACHE  = 196)        5938         530   91.8%
    Directory LRU         (ACP_DINDXCACHE=  49)        5350         232   95.8%
    FID Cache             (ACP_FIDCACHE  =  64)         303          12   96.1%
    Extent Cache          (ACP_EXTCACHE  =  64)         547          31   94.6%
    Quota Cache           (ACP_QUOCACHE  = 100)           0           0    0.0%

    Volume Synch Locks              2140      Window Turns                    353
    Volume Synch Locks Wait           27      Currently Open Files            254
    Dir/File Synch Locks            12389     Total Count of OPENs           2061
    Dir/file Synch Locks Wait           8     Total Count of ERASE QIOs       314
    Access Locks                    7502
    Free Space Cache Wait              4

    RMS GblBufQuo Remaining         1023      RMS_GBLBUFQUO (SYSGEN)          1024
    Global Pagefile Quota           1011      GBLPAGFIL (SYSGEN) Limit        1024
```

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         FFFFFFFF7FF91D58
Stack Limits (low)    FFFFFFFF7FF90000
             (high)   FFFFFFFF7FF92000
```

CLUE STACK identifies and displays the current stack together with the upper and lower stack limits.

2.
```
SDA> CLUE STACK
Stack Decoder:
--------------
System Stack (NULL Process):
Stack Pointer         FFFFFFFF887DFB28
Stack Limits (low)    FFFFFFFF887DE000
             (high)   FFFFFFFF887E0000

INVEXCEPTN Stack:
-----------------
Stack Pointer  SP =>  FFFFFFFF887DFB28

Information saved by Bugcheck:
a(Signal Array)       FFFFFFFFF887DFB28  FFFFFFFF 887DFCE8
```

```
Fixed Exception Context Area:
Linkage Pointer       FFFFFFFF887DFB30  FFFFFFFF 8007FA14  EXE$ALTQUEPKT_C+00044
a(Signal Array)       FFFFFFFF887DFB38  00000000 000001B8
a(Mechanism Array)    FFFFFFFF887DFB40  00000000 00000050
a(Exception Frame)    FFFFFFFF887DFB48  00000000 00000210  UCB$N_RSADDR
Exception FP          FFFFFFFF887DFB50  FFFFFFFF 80B93380
Unwind SP             FFFFFFFF887DFB58  00000000 00000000
Reinvokable FP        FFFFFFFF887DFB60  EEEEEEEE EEEEEEEE
Unwind Target         FFFFFFFF887DFB68  FFFFFFFF 887DFC60
#Sig Args/Byte Cnt    FFFFFFFF887DFB70  80B880E0 00000250  BUG$_NETRCVPKT
a(Msg)/Final Status   FFFFFFFF887DFB78  80B93380 00000001

Mechanism Array:
Flags/Arguments       FFFFFFFF887DFB80  00000000 0000002B
a(Establisher FP)     FFFFFFFF887DFB88  FFFFFFFF 887DFFB0
reserved/Depth        FFFFFFFF887DFB90  FFFFFFFF FFFFFFFD
a(Handler Data)       FFFFFFFF887DFB98  FFFFFFFF 808A8FD0  NETDRIVER_NPRW+023D0
a(Exception Frame)    FFFFFFFF887DFBA0  FFFFFFFF 887DFD40
a(Signal Array)       FFFFFFFF887DFBA8  FFFFFFFF 887DFCE8
saved R0              FFFFFFFF887DFBB0  FFFFFFFF 88EA1D60
saved R1              FFFFFFFF887DFBB8  00000000 00000001
saved R16             FFFFFFFF887DFBC0  00000000 00000000
saved R17             FFFFFFFF887DFBC8  00000000 00000000
saved R18             FFFFFFFF887DFBD0  FFFFFFFF 80859468  CACHE$GL_LRU_TIME
saved R19             FFFFFFFF887DFBD8  00000000 00000008
saved R20             FFFFFFFF887DFBE0  FFFFFFFF 80A0C4B6  SISR+0006E
saved R21             FFFFFFFF887DFBE8  00000000 0000003E
saved R22             FFFFFFFF887DFBF0  00000000 000017C7  CTL$C_CLIDATASZ+00227
saved R23             FFFFFFFF887DFBF8  00000000 000017C7  CTL$C_CLIDATASZ+00227
saved R24             FFFFFFFF887DFC00  00000000 00000000
saved R25             FFFFFFFF887DFC08  00000000 00000001
saved R26             FFFFFFFF887DFC10  FFFFFFFF 88EA0960
saved R27             FFFFFFFF887DFC18  00000000 0000FFFF
saved R28             FFFFFFFF887DFC20  00000000 00000001
FP Regs not valid     [......]
SP Align = 08(hex)    [......]

Signal Array:
Arguments             FFFFFFFF887DFCE8            00000005
Condition             FFFFFFFF887DFCEC            0000000C
Argument #2           FFFFFFFF887DFCF0            00000000
Argument #3           FFFFFFFF887DFCF4            00000000
Argument #4           FFFFFFFF887DFCF8            80125AC8  CACHE$TRUNCATE_C+00448
Argument #5           FFFFFFFF887DFCFC            00000804  UCB$M_VALID+00004


Exception Record:
Count/Flag/Kind       FFFFFFFF887DFD00  00000002 00000001
Expt Value            FFFFFFFF887DFD08  00000000 0000000C
Expt Next             FFFFFFFF887DFD10  00000000 00000000
Expt PC               FFFFFFFF887DFD18  FFFFFFFF 80125AC8  CACHE$TRUNCATE_C+00448
Extent/Kind           FFFFFFFF887DFD20  00000008 00000000
Value/Pointer         FFFFFFFF887DFD28  00000000 00000000
Extent/Kind           FFFFFFFF887DFD30  00000008 00000000
Value/Pointer         FFFFFFFF887DFD38  00000000 00000000

Interrupt/Exception Frame:
saved R2              FFFFFFFF887DFD40  00000000 00000280  BUG$_NOBUFPCKT
saved R3              FFFFFFFF887DFD48  00000000 00000000
saved R4              FFFFFFFF887DFD50  FFFFFFFF 80B774C0
saved R5              FFFFFFFF887DFD58  00000000 00000010
saved R6              FFFFFFFF887DFD60  00000000 00000200  IRP$M_TERMIO
saved R7              FFFFFFFF887DFD68  FFFFFFFF 88EA1D40
saved PC              FFFFFFFF887DFD70  FFFFFFFF 80125AC8  CACHE$TRUNCATE_C+00448
saved PS              FFFFFFFF887DFD78  10000000 00000804  IPL INT CURR PREV
SP Align = 10(hex)    [......]                    08  1  Kern Kern
```

```
Stack (not decoded):
                    FFFFFFFF887DFD90  FFFFFFFF 80125840  CACHE$TRUNCATE_C+001C0
                    FFFFFFFF887DFD98  FFFFFFFF 81609488
                    FFFFFFFF887DFDA0  FFFFFFFF 80D61AC0
                    FFFFFFFF887DFDA8  FFFFFFFF 80859990  CACHE$TRUNCATE+00020
                    FFFFFFFF887DFDB0  FFFFFFFF 8012631C  CACHE$TRUNCATE_C+00C9C
                    FFFFFFFF887DFDB8  FFFFFFFF 80B9B7C0
                    FFFFFFFF887DFDC0  FFFFFFFF 80D61AC0
                    FFFFFFFF887DFDC8  00000000 00000200  IRP$M_TERMIO
                    FFFFFFFF887DFDD0  FFFFFFFF 8009E370  SCH$INTERRUPT+003D0
                    FFFFFFFF887DFDD8  00000000 00000000
                    FFFFFFFF887DFDE0  00000000 0000029F  BUG$_NONEXSTACP+00007
                    FFFFFFFF887DFDE8  00000000 00000001
                    FFFFFFFF887DFDF0  FFFFFFFF 80859A00  CACHE$TRUNCATE+00090
                    FFFFFFFF887DFDF8  FFFFFFFF 80804200  EXE$GR_SYSTEM_DATA_CELLS
                    FFFFFFFF887DFE00  FFFFFFFF 800273E0  EXE_STD$QUEUE_FORK_C+0027
                    FFFFFFFF887DFE08  00000004 80AF4940
                    FFFFFFFF887DFE10  FFFFFFFF 80B11D00
                    FFFFFFFF887DFE18  FFFFFFFF 801240E8  CACHE$IOPOST_C+00618
                    FFFFFFFF887DFE20  00000000 00000001
                    FFFFFFFF887DFE28  00000000 7FF9D190
                    FFFFFFFF887DFE30  00000000 00000000
                    FFFFFFFF887DFE38  FFFFFFFF 80859810  CACHE$IOPOST
                    FFFFFFFF887DFE40  FFFFFFFF 80072EC8  IOC$IOPOST_C+00248
                    FFFFFFFF887DFE48  00000000 00000280  BUG$_NOBUFPCKT
                    FFFFFFFF887DFE50  FFFFFFFF 81609488
                    FFFFFFFF887DFE58  FFFFFFFF 80B9B7C0
                    FFFFFFFF887DFE60  00000000 00000200  IRP$M_TERMIO
                    FFFFFFFF887DFE68  FFFFFFFF 8009E370  SCH$INTERRUPT+003D0
                    FFFFFFFF887DFE70  00000000 00000001
                    FFFFFFFF887DFE78  00000000 00000000
                    FFFFFFFF887DFE80  00000000 7FF9D190
                    FFFFFFFF887DFE88  00000000 00000000
                    FFFFFFFF887DFE90  FFFFFFFF 8083A510  IOC$IOPOST
                    FFFFFFFF887DFE98  00000000 00000001
                    FFFFFFFF887DFEA0  00000000 7FF9D190
                    FFFFFFFF887DFEA8  00000000 00000000
                    FFFFFFFF887DFEB0  00000000 00000001
                    FFFFFFFF887DFEB8  FFFFFFFF 808407A0  SCH$IDLE
                    FFFFFFFF887DFEC0  FFFFFFFF 80804200  EXE$GR_SYSTEM_DATA_CELLS
                    FFFFFFFF887DFEC8  00000000 00000001
                    FFFFFFFF887DFED0  00000000 00000303  UCB$Q_PI_IQ+00003
                    FFFFFFFF887DFED8  00000000 0000000D
                    FFFFFFFF887DFEE0  FFFFFFFF 808319A0  SYS$CPU_ROUTINES_0302_NPRW+05
                    FFFFFFFF887DFEE8  FFFFFFFF 80804200  EXE$GR_SYSTEM_DATA_CELLS
                    FFFFFFFF887DFEF0  FFFFFFFF 80A0C4B6  SISR+0006E
                    FFFFFFFF887DFEF8  FFFFFFFF 80A0C4B6  SISR+0006E
                    FFFFFFFF887DFF00  00000000 00000200  IRP$M_TERMIO
                    FFFFFFFF887DFF08  00000000 00000037
                    FFFFFFFF887DFF10  00000000 00000201  UCB$W_LMERRCNT+00001
                    FFFFFFFF887DFF18  FFFFFFFF 80804200  EXE$GR_SYSTEM_DATA_CELLS
                    FFFFFFFF887DFF20  00000000 7C46FD50
                    FFFFFFFF887DFF28  00000000 00000008
                    FFFFFFFF887DFF30  FFFFFFFF 80804200  EXE$GR_SYSTEM_DATA_CELLS
                    FFFFFFFF887DFF38  FFFFFFFF 887DFFB0
                    FFFFFFFF887DFF40  FFFFFFFF 80840064  SCH$AL_CPU_PRIORITY+000E4
                    FFFFFFFF887DFF48  FFFFFFFF 80842CE0  SCH$GR_SCHEDULER_LINKAGE_SEC
                    FFFFFFFF887DFF50  FFFFFFFF 80CC7C00
                    FFFFFFFF887DFF58  00000000 00000001
                    FFFFFFFF887DFF60  FFFFFFFF 80A0C000
                    FFFFFFFF887DFF68  FFFFFFFF 8009E370  SCH$INTERRUPT+003D0
                    FFFFFFFF887DFF70  FFFFFFFF 80090ABC  SCH$IDLE_C+0009C
                    FFFFFFFF887DFF78  30000000 00000303  UCB$Q_PI_IQ+00003
                    FFFFFFFF887DFF80  00000000 00000001
                    FFFFFFFF887DFF88  00000003 00000008
```

```
                      FFFFFFFF887DFF90  FFFFFFFF 80090A90  SCH$IDLE_C+00070
                      FFFFFFFF887DFF98  00000000 00000001
                      FFFFFFFF887DFFA0  FFFFFFFF 80840064  SCH$AL_CPU_PRIORITY+000E4
                      FFFFFFFF887DFFA8  00000000 00000001

Stack Frame:
PV                    FFFFFFFF887DFFB0  FFFFFFFF 808407A0  SCH$IDLE
        Entry Point             FFFFFFFF 80090A20  SCH$IDLE_C
                      FFFFFFFF887DFFB8  00000000 00000000
                      FFFFFFFF887DFFC0  FFFFFFFF 80CC7C00
return PC             FFFFFFFF887DFFC8  FFFFFFFF 8009E094  SCH$INTERRUPT+000F4
saved R2              FFFFFFFF887DFFD0  FFFFFFFF 80840064  SCH$AL_CPU_PRIORITY+000E4
saved R4              FFFFFFFF887DFFD8  FFFFFFFF 80CC7C00
saved R13             FFFFFFFF887DFFE0  00000000 7FEA8850
saved R14             FFFFFFFF887DFFE8  00000005 00000000
saved R15             FFFFFFFF887DFFF0  00000000 00000001
saved FP              FFFFFFFF887DFFF8  00000000 7FE1FA30
```

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.

/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