z/VM
7.4

Dump Viewing Facility

GC24-6284-74

Note:

Before you use this information and the product it supports, read the information in "Notices" on page 193.

This edition applies to version 7, release 4 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2024-09-18

Contents

Figures

Tables

About This Document

This document is designed to assist those who use the IBM® z/VM® Dump Viewing Facility in performing problem determination (PD) and problem source identification (PSI). It provides a description of the Dump Viewing Facility usage information and reference material. The Dump Viewing Facility analyzes and manages system software problems interactively under the conversational monitor system (CMS).

Intended Audience

This document is provided to assist system programmers, service personnel, and those who use the Dump Viewing Facility to analyze dump data in order to perform problem determination (PD) and problem source identification (PSI).

This information is written for people who have experience with basic debugging techniques. An understanding of the z/VM components and other licensed programs is also helpful.

Note: The VM Dump Tool is the only supported way to look at a CP dump, but can also be used to look at VMDUMPs of other components of VM. See *z/VM: VM Dump Tool*, for more information.

Syntax, Message, and Response Conventions

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The ►►—— symbol indicates the beginning of the syntax diagram.
- The ——► symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.
- The ►—— symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.
- The ——►◄ symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in Table 1 on page xiii.

Table 1. Examples of Syntax Diagram Conventions

Syntax Diagram Convention	Example
Keywords and Constants A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown. In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.	►►— KEYWORD —►◄

Table 1. Examples of Syntax Diagram Conventions (continued)

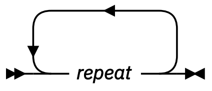
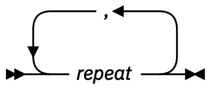
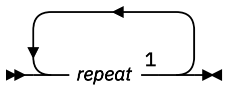
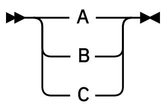
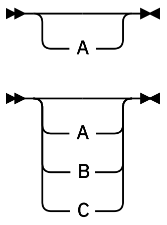
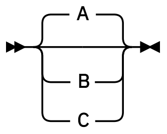
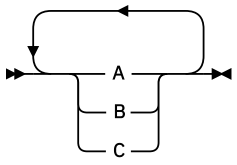
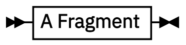
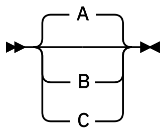
Syntax Diagram Convention	**Example**
Abbreviations Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated. In this example, you can specify KEYWO, KEYWOR, or KEYWORD.	►► KEYWOrd ►◄
Symbols You must specify these symbols exactly as they appear in the syntax diagram.	***** Asterisk **:** Colon **,** Comma **=** Equal Sign **-** Hyphen **()** Parentheses **.** Period
Variables A variable appears in highlighted lowercase, usually italics. In this example, *var_name* represents a variable that you must specify following KEYWORD.	►► KEYWOrd — *var_name* ►◄
Repetitions An arrow returning to the left means that the item can be repeated. A character within the arrow means that you must separate each repetition of the item with that character. A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated. Syntax notes may also be used to explain other special aspects of the syntax.	   Notes: 1 Specify *repeat* up to 5 times.
Required Item or Choice When an item is on the line, it is required. In this example, you must specify A. When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.	►► A ►◄ 

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	**Example**
Optional Item or Choice When an item is below the line, it is optional. In this example, you can choose A or nothing at all. When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.	
Defaults When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line. In this example, A is the default. You can override A by choosing B or C.	
Repeatable Choice A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item. In this example, you can choose any combination of A, B, or C.	
Syntax Fragment Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name. In this example, the fragment is named "A Fragment."	 **A Fragment** 

Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

xxx
: Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.

[ ]
: Brackets enclose optional text that might be displayed.

{ }
: Braces enclose alternative versions of text, one of which will be displayed.

|
: The vertical bar separates items within brackets or braces.

...
: The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

Using the Online HELP Facility

You can receive online information about the commands described in this book using the z/VM HELP Facility. For example, to display a menu of DUMPVIEW commands, enter:

```
help dumpview menu
```

To display information about a specific DUMPVIEW command (ADDMAP in this example), enter:

```
help dumpview addmap
```

You can also display information about a message by entering one of the following commands:

```
help msgid or help msg msgid
```

For example, to display information about message HCSDSS200I, you can enter one of the following commands:

```
help hcsdss200i or help hcs200i or help msg hcs200i
```

For more information about using the HELP Facility, see the *z/VM: CMS User's Guide*. To display the main HELP Task Menu, enter:

```
help
```

For more information about the HELP command, see the *z/VM: CMS Commands and Utilities Reference* or enter:

```
help cms help
```

Where to Find More Information

For information about related publications, see the "Bibliography" on page 197.

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See How to send feedback to IBM for additional information.

Summary of Changes for z/VM: Dump Viewing Facility

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line (**|**) to the left of the change.

SC24-6284-74, z/VM 7.4 (September 2024)

This edition supports the general availability of z/VM 7.4. Note that the publication number suffix (-74) indicates the z/VM release to which this edition applies.

SC24-6284-73, z/VM 7.3 (December 2023)

This edition includes terminology, maintenance, and editorial changes.

SC24-6284-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

GC24-6284-02, z/VM 7.2 (September 2020)

This edition supports the general availability of z/VM 7.2.

Chapter 1. Introduction

The Dump Viewing Facility analyzes and manages system software problems interactively under the conversational monitor system (CMS).

Major Functions

The Dump Viewing Facility assists in the following tasks:

- Interactively analyzing dump data
- Formatting and printing dump data
- Reducing trace tables created by trace service tools
- Recognizing duplicate problems

Interactively Analyzing Dump Data

The Dump Viewing Facility provides a variety of commands and subcommands that allow the user to interactively locate and display dump data. Use the Dump Viewing Facility to do the following:

- Display real program status words, registers, clocks, and the timer
- Display formatted data from any z/VM control block or data area
- Display data in hexadecimal and EBCDIC
- Display data using 24- or 31-bit indirect addressing
- Display a chain of control block addresses in hexadecimal or display the data within the control blocks
- Locate a hexadecimal or EBCDIC string in the dump
- Print output from any DUMPSCAN subcommand
- Determine a module entry point and displacement, given an address
- Determine an address, given a module or entry-point name
- Scroll forward or backward while viewing hexadecimal data
- Create a load map of module names and their entry points with addresses and displacements
- Assign symbolic names to subcommands
- Format, reduce, and scroll through trace tables within a dump dumps

Formatting and Printing Dump Data

The Dump Viewing Facility uses a dump file created by the DUMPLOAD utility to print summary reports. The summary reports available are as follows:

- Module load maps
- Symptom records
- General processor information
- Dump ID (for virtual machine dumps only)

Using the PRTDUMP command, you can print all pages of dumped storage in hexadecimal.

Recognizing Duplicate Problems

The Dump Viewing Facility provides the VIEWSYM command to help you identify duplicate problems. Whenever a dump is requested, a symptom record is created. A symptom recording virtual machine can retrieve these symptom records and place them in a repository. You can then use the VIEWSYM command to examine the repository for duplicate symptom records.

Types of Dumps the Dump Viewing Facility Processes

You can use the Dump Viewing Facility to process any of the following kinds of dumps:

- VM (virtual machine) dumps include the following:
  - CMS (conversational monitor system)
  - GCS (group control system)
  - Pass-Through. In this document, Pass-Through refers to both the VM/Remote 3270 Display Option and the Pass-Through Virtual Machine (PVM).
  - RSCS (Remote Spooling Communications Subsystem)
  - SFS (shared file system)

    Note: The Coordinated Resource Recovery (CRR) server uses SFS facilities to take a dump. All discussions of SFS dumps in this manual therefore include CRR dumps.
  - TSAF (transparent services access facility).

For information about processing CP dumps, see *z/VM: VM Dump Tool*.

Requirements for Using the Dump Viewing Facility

To use all the functions the Dump Viewing Facility provides to examine a dump, you need the following:

- The Dump Viewing Facility installed on your system.

  For more information about installing the Dump Viewing Facility, see *z/VM: Installation Guide*.
- A copy of the dump or dumps you want to examine that have been processed by DUMPLOAD.
- In some instances, a copy of the load map that describes the system from which your dump was taken.

  For more information on when the Dump Viewing Facility requires a load map and how one is created and processed, see "Using Load Maps" on page 9.

Storage Requirements

The disk storage requirements for the Dump Viewing Facility include space for the dump and the load map when it is used.

Table 2 on page 2 shows the space requirements in cylinders for CKD/ECKD™ devices or blocks for FBA devices for the CMS file containing the load map.

Table 2. Space Requirements in Cylinders & Blocks for the Load Map File

BLKSIZE	3380	3390	FBA
512	16	15	23,040
1024	14	13	18,720
2048	12	10	14,400
4096	11	9	12,960

After processing with the Dump Viewing Facility MAP command, the Dump Viewing Facility module map fits on 1 cylinder of CKD/ECKD devices or 1440 blocks of FBA disk space.

Table 3 on page 3 shows the space requirements in cylinders for CKD/ECKD devices or blocks for FBA devices for each 16 MB of dumped storage with the Dump Viewing Facility module map appended dynamically or appended by the ADDMAP command. (For DASD types other than 3380 or 3390, you must calculate an equivalent amount of space.)

Table 3. Space Requirements in Cylinders & Blocks Per 16 MB of Dumped Storage

BLKSIZE	3380	3390	FBA
512	34	36	52,700
1024	36	34	48,960
2048	31	27	38,880
4096	28	23	33,120

Causes for a Dump

In z/VM, dumps can be initiated by hardware, software, or a user. The cause of the dump determines the type of dump: CP, virtual machine, or stand-alone. For instance, if a machine check occurs, a CP dump results. If a user enters the CP VMDUMP command in a CMS virtual machine, a virtual machine dump results. For more information on CP dump, see *z/VM: VM Dump Tool*.

Note: If CP is unable to take an abend dump, you can initiate a stand-alone dump.

To take a stand-alone dump, use the stand-alone dump utility. For more information on creating the stand-alone dump utility, see *z/VM: CP Planning and Administration*, and for more information on running the stand-alone dump utility program see *z/VM: System Operation*.

Hardware-Initiated Dumps

Not all hardware errors result in a dump being taken. Some examples of hardware errors that do result in dumps are:

- Machine checks in the central processor
- Storage checks in main storage
- Channel checks in the I/O channels

Some hardware errors cause a dump to be taken immediately as a result of a machine check condition. Other errors may alter the condition of the hardware (for example, a processor or main storage) in a manner that eventually will cause CP to detect an abnormal condition and take an abend dump.

If a dump is taken, it may contain symptoms of the hardware error. You may find additional symptoms by examining the hardware error log in the error-recording cylinders, using the environmental recording, editing, and printing (EREP) facility, and by examining messages sent to the system operator's console.

Software-Initiated Dumps

Generally, a software error occurs when a sequence of instructions executed by a processor results in a condition that is incompatible with the design of the software system. Software errors have a variety of symptoms. Some typical symptoms are

Symptom	Software Error
Loop	A sequence of instructions is executed over and over again, infinitely.
Wait	The software system cannot find work to do, or it encounters a condition that cannot be resolved.
Lockout	A user ID has become disabled, nondispatchable, or has no work-tasks to be run.
Storage overlay	A program has stored data in the wrong location in real storage.
Invalid data	A system control block or data area contains data that is inconsistent or erroneous.

Invalid address
: A control block or data area contains an address that is outside the storage areas to which the program has access; or, a control block or data area points to a nonexistent control block. Because some addresses are generated by programs from data in control blocks, an invalid address can be generated if the data is in error.

Invalid instructions
: An instruction is found that does not conform to the system architecture, possibly because it has been modified in main storage.

Invalid addresses and instructions result in program checks. The other kinds of software errors are usually detected by CP as abnormal conditions. Both result in abnormal termination of CP and a dump being taken.

User-Initiated Dumps

System Restarts

A system restart is typically initiated by the system operator and results in a dump being taken. The system is usually restarted when a problem in the hardware or software results in a wait, loop, or lockout of one or more important user IDs, or in degraded performance. In these situations, it is generally desirable to reinitialize CP. System restart can perform this reinitializing and capture the circumstances of the problem in the dump.

VMDUMP Command

The CP command, VMDUMP, produces a dump of all or selected pages of storage that appear real to your virtual machine (second-level storage), including VM data spaces. In order for the resulting dump to be usable by the Dump Viewing Facility, you must use the DUMPLOAD command to load the dump into a CMS file. The dump includes selected information for the virtual processor on which you entered the VMDUMP command.

SNAPDUMP Command

The CP command, SNAPDUMP, will produce a dump of the entire z/VM system and is identical, in format, to a hard abend dump but will not result in system termination. This type of dump might be especially helpful when trying to debug a "hung user" type of problem or when it is impossible to shut the system down for dump generation and analysis. The SNAPDUMP command value settings can be altered by the CP SET ABEND and SET DUMP commands. For more information on the exact syntax of the SNAPDUMP and VMDUMP commands, see *z/VM: CP Commands and Utilities Reference*.

Location of a Dump

The dump taken is sent to a tape or to the virtual reader of a specific virtual machine (user ID). If a dump is to be analyzed using the Dump Viewing Facility, it must be directed to tape or to a virtual reader. The destination of the dump is determined by using the CP SET DUMP command. For more information on the SET DUMP command, see *z/VM: CP Commands and Utilities Reference*.

Use of Dump Information

You can use data from a dump to help locate the source of the problem that caused the dump. As previously discussed, in z/VM a dump may result from any of the following:

- A hardware error
- A software error
- The system operator initiating a system restart
- A user issuing either the CP SNAPDUMP or VMDUMP commands

Use the following two main steps to narrow your search for the cause of the dump:

Problem determination
Finding out whether the problem is caused by hardware or software.

Problem source identification
Isolating the problem in a particular component of the software system.

After you locate the error in the software, you can use error messages and the symptom record to refine your analysis of the problem further.

Problem Determination

The goal of problem determination is to discover whether the dump was the result of a hardware or software error. Sometimes the clues are obvious. For example, if a machine check, channel check, or storage check preceded the dump, the problem is likely to be a hardware error. If the dump is a CP abend dump, the cause is more likely to be a software error. When a restart dump has been taken, you will probably have to examine the conditions of both the hardware and software to make the determination.

After you have determined whether the problem is hardware or software, problem determination is complete. The discussion in this document focuses on problem source identification for software.

Problem Source Identification

Problem source identification is the second step in analyzing a software problem. It consists of isolating the cause of a problem to a particular component of the software system. z/VM has eight components:

- Control program (CP)
- Conversational monitor system (CMS)
- Service EXECs (VMSES/E)
- Dump Viewing Facility
- Procedures language (VM/REXX)
- Group control system (GCS)
- Transparent services access facility (TSAF)
- APPC/VM VTAM® support (AVS)

In addition to these components, there are several program products that can run in the z/VM environment. Problem source identification is complete when you have determined the particular component or program in which the error occurred.

Using Error Messages

One method of determining where the problem occurred is to examine any error messages. These messages usually identify the immediate cause of the dump. For example, a CP abend dump is identified as such by a message. CMS and the Dump Viewing Facility also issue messages when they detect errors. Similar messages may be sent to the system operator's console. You can look at CP, CMS, and Dump Viewing Facility messages in *z/VM: Other Components Messages and Codes*. The message descriptions identify the failing component and briefly describe the error conditions encountered.

If a message indicates that an error occurred in CP, you can use the message code to determine which module in CP encountered the error. The scenarios in this chapter describe this in more detail.

Using the Symptom Record to Identify Duplicate Problems

When a virtual machine is dumped by the VMDUMP command, a symptom record is created and included in the dump. In addition, a copy of this symptom record is sent to the Symptom Record Recording virtual machine. The symptom record summarizes data about the state of the system when the dump was taken. The Dump Viewing Facility can format the symptom record for display and printing. For additional information on symptom records, see the SYMPTOM subcommand in Chapter

5, “DUMPSCAN Subcommand Reference,” on page 51, and the VIEWSYM command in Chapter 3, “Command Reference,” on page 21.

You can use the keywords and formatted data from the symptom record to determine whether the problem has occurred on your system before. You can compare the symptom record data in a new dump to symptom records in dumps that already exist, keyword by keyword. A match on all the data indicates that the new problem may be a duplicate.

You can identify duplicate problems by using the VIEWSYM command to search the repository of symptom records. You can also ask the IBM Support Center to search the IBM database. This database contains information about all the problems reported to IBM by z/VM users.

Commands Associated with the Dump Viewing Facility

You can use the following commands with the Dump Viewing Facility: ADDMAP, DUMPSCAN, MAP, PRTDUMP, and VIEWSYM.

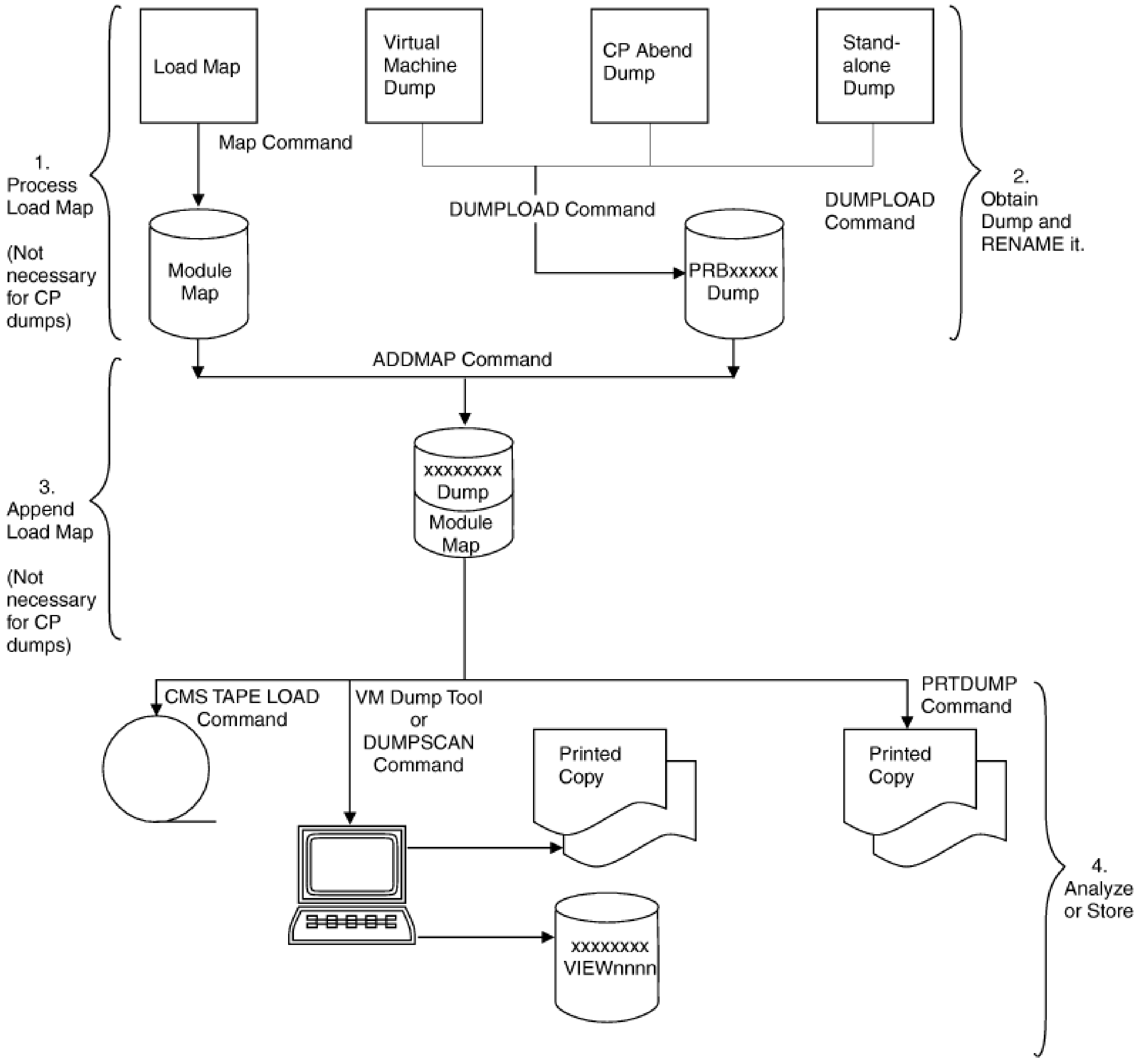


Figure 1. Command Structure for the Dump Viewing Facility's ADDMAP, DUMPSCAN, MAP, and PRTDUMP Commands

Note: The following information on the MAP and ADDMAP commands is required for non-CP dumps. For more information on when you do and do not need to use the MAP and ADDMAP commands, see "Creating Load Maps" on page 9.

1. The MAP command compresses the z/VM load map created at system generation time into a format that the Dump Viewing Facility can process. The compressed module map correlates module and entry-point names with addresses in the dump.
2. Use the ADDMAP command to append the compressed module map to the CMS file containing the dump processed by the DUMPLOAD command. For more information on the DUMPLOAD command, see *z/VM: CP Commands and Utilities Reference*.
3. Use the DUMPSCAN command to interactively view virtual machine dumps. The DUMPSCAN subcommands are described in Chapter 5, "DUMPSCAN Subcommand Reference," on page 51 of this manual. Use the VM Dump Tool to process CP and stand-alone dumps.
4. Use the PRTDUMP command to print the summary reports available through the Dump Viewing Facility. Each summary report contains data the problem solver most frequently requires.

   PRTDUMP can also print dumped storage.
5. Use the VIEWSYM command to help you identify duplicate problems. Whenever a dump is requested, a symptom record is created. A symptom-recording virtual machine can retrieve these symptom records and place them in a repository. You can then use the VIEWSYM command to examine the repository for duplicate symptom records.

   Besides searching for duplicate symptom records, the VIEWSYM command also permits you to create a summary list of symptom records and to examine individual symptom records.

Figure 1 on page 6 illustrates the command structure for the ADDMAP, DUMPSCAN, MAP, and PRTDUMP commands. Figure 2 on page 7 illustrates the command structure for the VIEWSYM command.

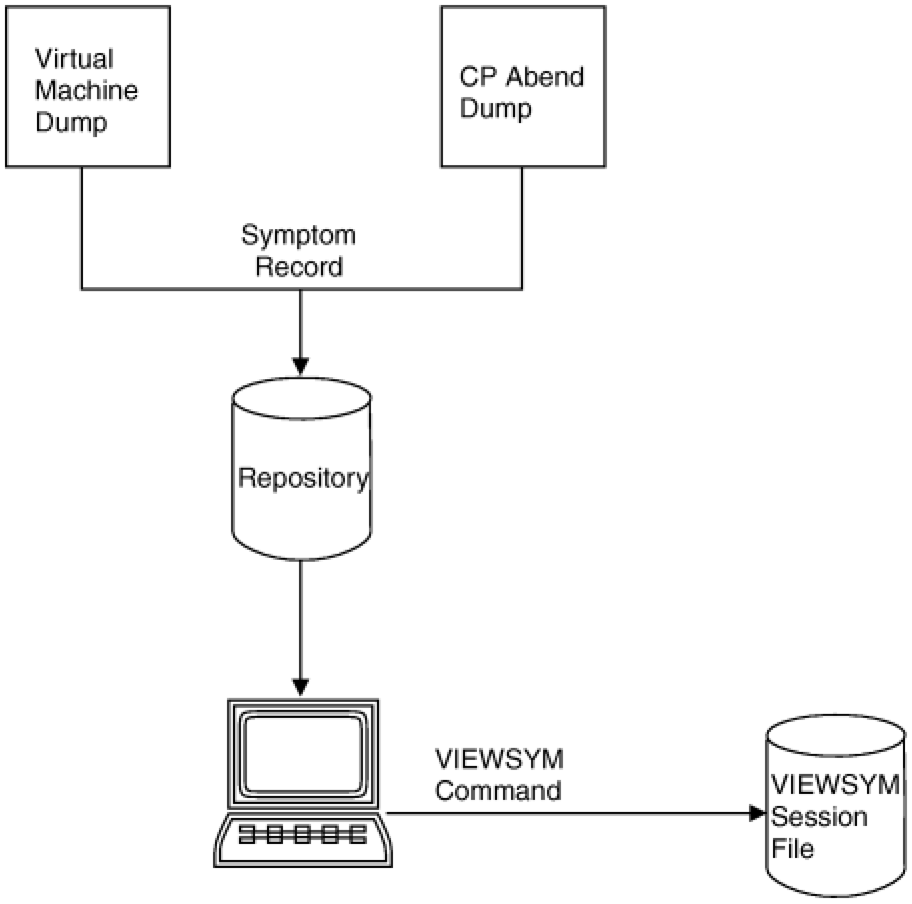


Figure 2. Command Structure for the Dump Viewing Facility's VIEWSYM Command

Servicing the Dump Viewing Facility

Corrective and preventive maintenance of the Dump Viewing Facility is performed using standard z/VM procedures. For more information on these maintenance procedures, see *z/VM: Service Guide*.

Writing Dump Data to Tape

Use the following commands to write dump data to tape:

1. Enter the DUMPLOAD command to place the dump on your A-disk.
2. Rename the dump file name with the CMS RENAME command. For example, when you enter the DUMPLOAD command, the dump file can be named PRB00001 DUMP0001 A. You may rename this file by issuing the following command:

   ```
   rename prb00001 dump0001 a cpdump01 dump0001 A
   ```

3. Finally, copy the dump to tape using the CMS TAPE DUMP command. The format for this command is:

   ```
   tape dump filename filetype filemode
   ```

For more information on the CMS commands, see *z/VM: CMS Commands and Utilities Reference*.

Chapter 2. Usage Guide

This chapter describes how to create and use different dumps.

Preparing a Dump for Use with the Dump Viewing Facility

If you wish to use the Dump Viewing Facility to analyze a dump, you first have to prepare the dump by following these steps:

1. Log on with the appropriate user ID.

   This user ID must:

   - Be the user ID to which the dump was sent, if the dump was sent to a virtual reader
   - Have enough unused space on its A-disk to hold the dump file.

     For the amount of storage required for dumps, see the “Storage Requirements” on page 2
2. Use the DUMPLOAD command to load the dump into a CMS file.

   Issuing the DUMPLOAD command puts the dump in a CMS file on the A-disk, PRB*xxxxx* DUMP*nnnn* A, where *xxxxx* is a number from 00000 to 99999, and *nnnn* is a number from 0001 to 9999, depending on the IDs of the dump files already on the A-disk.

   If you wish to print summary information about a dump, use the PRTDUMP command (see “PRTDUMP Command” on page 29).
3. If dump files PRB00000 through PRB99999 already exist, DUMPLOAD erases all the PRB00000 files and uses the file name for the current dump. If you wish, rename the dump file by using the CMS RENAME command.

   For more information on the CMS RENAME command, see *z/VM: CMS Commands and Utilities Reference*.

   When renaming the file, keep the general format of *filename* DUMP*nnnn* A. The Dump Viewing Facility allows file names in the form *xxxxxxxx*, where *xxxxxxxx* is a 1- to 8-character string that may consist of the characters 0-9, A-Z, @ (at sign), # (pound sign), - (hyphen), _ (underscore), + (plus sign), : (colon), and $ (dollar sign).
4. Create and append a module map to the dump you want to examine as follows:

   - Use the MAP and ADDMAP commands.

     The MAP command creates a module map from a load map. The ADDMAP command appends the newly created module map to the dump. For additional information on using of these commands, see “Creating Load Maps” on page 9 and “Creating Module Maps” on page 10 in this chapter.

Using Load Maps

The Dump Viewing Facility does not dynamically generate module maps for virtual machine dumps. Therefore, it requires a virtual machine load map.

Creating Load Maps

You should create and save a load map whenever you generate a new system. This load map is required for generation of the Dump Viewing Facility module map for virtual machine dumps.

Virtual Machine Load Maps Table 4 on page 10 identifies the publication that describes the procedure for creating the specific virtual machine load map.

Table 4. Instructions for Creating Virtual Machine Load Maps

Load Map	Reference
CMS	*z/VM: Service Guide*
GCS	*z/VM: Service Guide*
PVM	*VM/SP™ Pass-Through Guide and Reference*.
RSCS	*z/VM: RSCS Networking Diagnosis*

Note: SFS (including CRR), AVS, and TSAF do not have nuclei, so you cannot create load maps for them. Consult the CMS load map.

Creating Module Maps

1. For virtual machine dumps, use the following two-step method, which employs the MAP and ADDMAP commands with the required virtual machine load map(s):
   a. Use the MAP command to convert the load map into a module map.
   b. Use the ADDMAP command to append the module map to the dump you specify in the command. For more information on using these commands, see Chapter 3, "Command Reference," on page 21.

      This method requires that the dump type is supported by an entry in the HCSTAB table. Entries can be added to this table, or existing entries can be modified, by using the procedure described in Appendix A, "Using Attachment Interfaces," on page 139 under "Modifying the HCSTAB Table" on page 139.

The dump file is now ready for viewing by using the Dump Viewing Facility.

Viewing Dumps

To view a dump, use the Dump Viewing Facility DUMPSCAN command. For example, if the dump you wish to analyze is in the file named HUNGUSER DUMP0001 A, you can enter the command:

```
dumpscan hunguser
```

When the dump file has been accessed, you are notified by the READY status message. You are in an XEDIT environment ready to enter DUMPSCAN subcommands on the command line.

Use the subcommands of the DUMPSCAN command to view the data in the dump file. These subcommands are explained in Chapter 5, "DUMPSCAN Subcommand Reference," on page 51.

Some types of dumps have component-unique DUMPSCAN subcommands. Table 5 on page 10 shows where to find information about using these subcommands.

Table 5. Using Component-Unique DUMPSCAN Subcommands

Virtual Machine Dump Type	Reference
CMS	Chapter 5, "DUMPSCAN Subcommand Reference," on page 51
GCS	Chapter 5, "DUMPSCAN Subcommand Reference," on page 51
TSAF	Chapter 5, "DUMPSCAN Subcommand Reference," on page 51
AVS	Chapter 5, "DUMPSCAN Subcommand Reference," on page 51
SFS	Chapter 5, "DUMPSCAN Subcommand Reference," on page 51
PVM	*VM/SP Pass-Through Facility Logic*

Table 5. Using Component-Unique DUMPSCAN Subcommands (continued)

Virtual Machine Dump Type	Reference
RSCS	*z/VM: RSCS Networking Diagnosis*

Using the Session File

The Dump Viewing Facility uses full-screen XEDIT functions so that you can scroll back and forth through dump data in full-screen mode. You can scroll through any data previously viewed without reentering the command to display the data. You can edit your session file while in the Dump Viewing Facility. This annotation feature lets you make comments within a dump session file before passing the dump on to the next level of problem determination.

The dump viewing session can be filed on your A-disk with the XEDIT FILE subcommand. When you view the same dump later, the saved file is reactivated and the new session is appended to the *dumpname* VIEW*nnnn* file containing the previous sessions.

Virtual Machine Dumps in an XC Environment

Dumps of address spaces produce a separate dump file for each address space. All such related XC dump files have the same file name; their file types are numbered sequentially from "DUMP0001". (See the VMDUMP and DUMPLOAD commands in the *z/VM: CP Commands and Utilities Reference* for more details on how dumps are created for XC virtual machines and how the CMS dump files are named.)

You can use the ASID subcommand of DUMPSCAN to list all the related dumps (see "ASID Subcommand" on page 63). Output from the DUMPSCAN subcommands ASID and ACCLIST (which display XC architecture related information) is dependent on DUMPSCAN having access to these related address space dumps, therefore:

- The file types for these CMS dump files should not be changed.
- Changes to the file name should be made consistently for all the files. The Dump Viewing Facility uses the file name to relate files from the same dump. Conversely, you should not have duplicate file names for different dumps, even when the two sets of dump files are accessed at different file modes.

ASID and ACCLIST issue error messages when any dump file containing this special information cannot be accessed.

For XC virtual machine dumps only, the output from the SYMPTOM subcommand of DUMPSCAN displays a count of the total number of address spaces that were dumped by the CP VMDUMP command or by DIAGNOSE code X'94'. With this information, you can determine what other CMS dump files to look for, because the virtual machine dump files created by DUMPLOAD have sequentially numbered file types, beginning with DUMP0001.

Like the SYMPTOM subcommand, the VIEWSYM command displays a count of the number of address spaces dumped by the CP VMDUMP command or by DIAGNOSE code X'94'. See "VIEWSYM Command" on page 33 for the VIEWSYM Command Menu.

Use the CMS FILELIST command to locate dump files. Search on the file name to list the related dumps.

Viewing Several Dump Files at a Time

The DUMPSCAN session file has the same file name as the dump file. The file type is VIEWnnnn where nnnn is the number from the dump file type, DUMPnnnn.

The DUMPSCAN subcommand of the DUMPSCAN command works in a manner similar to that of the XEDIT subcommand of XEDIT. When the DUMPSCAN subcommand is issued with an operand specifying the name of an additional dump file to be viewed, the new file is added to the current DUMPSCAN command file ring, and the current screen is updated to reflect the new dump being viewed. If the user is in a split-screen mode, only the screen where the subcommand was issued is updated.

If the DUMPSCAN subcommand is entered with no operands, it switches to the next VIEWnnnn file in the current DUMPSCAN command file ring. If there are no other VIEWnnnn files, DUMPSCAN remains at the current dump file.

If the DUMPSCAN XEDIT subcommand is entered with an operand of XEDIT, then the next file in the XEDIT command file ring is displayed.

Now that the DUMPSCAN command and subcommand share the same name, the user needs to be aware of which DUMPSCAN has been issued when viewing multiple dumps. For example, if "CMS DUMPSCAN fn ft" is issued from a DUMPSCAN command line, then the DUMPSCAN command is invoked, resulting in the creation of a new DUMPSCAN command file ring that is independent of the existing ring. It is therefore possible to view the same dump from two separate file rings, which can lead to some confusion. If "DUMPSCAN fn ft" is issued from a DUMPSCAN command line, then the DUMPSCAN subcommand gets control and the new dump file is added to the existing DUMPSCAN ring if it does not already exist in the ring; otherwise, the DUMPSCAN subcommand positions the user at the existing dump view file.

Virtual Machine Dump Formats

The format, or "type", of a virtual machine dump (specified with the FORMAT option on the CP VMDUMP command) may not be known at the time the dump is taken. You may have to tell the Dump Viewing Facility the type of VMDUMP that is being viewed before these Dump Viewing Facility functions can be performed:

- Adding module maps via the ADDMAP command
- Invoking extraction and formatting routines
- Using the BLOCK subcommand table interface.

The FORMAT subcommand of DUMPSCAN allows the user to change the type of the virtual machine dump being viewed (see the description for the FORMAT subcommand in Chapter 5, "DUMPSCAN Subcommand Reference," on page 51). If the STORAGE option of the PRTDUMP command is used to print the storage from the dump, and if the format or type of virtual machine dump had previously been set either by the CP VMDUMP command or by a previous invocation of the DUMPSCAN FORMAT subcommand, then the dump type must be changed to the default value "FILE" by reissuing the FORMAT subcommand of DUMPSCAN prior to issuing PRTDUMP. To return to the original dump type environment, enter the FORMAT subcommand again and specify the desired virtual machine dump type.

Viewing Dumps of Licensed Programs and z/VM Features

The Dump Viewing Facility provides additional support for viewing data particular to specific virtual machine dumps. With this support, a user is able to find and format data in licensed programs' virtual machine dumps and display it using the Dump Viewing Facility.

The Dump Viewing Facility provides this support for dumps taken from the following:

- Conversational Monitor System (CMS)
- Data Facility Storage Management Subsystem for VM (DFSMS/VM)
- Group Control System (GCS)
- Pass-Through Virtual Machine (PVM)
- Remote Spooling Communications System (RSCS)

See Appendix A, "Using Attachment Interfaces," on page 139 for more information regarding this support.

Writing DUMPSCAN Macros

The DUMPSCAN subcommands help you to analyze dump data interactively, and in most situations there is a subcommand that will give you the needed data in a usable format. However, you can write macros to customize and automate the powers of DUMPSCAN, creating your own powerful tools to analyze dump data. By writing macros, you can:

- Expand the basic subcommand set

- Tailor the basic subcommand output for
  - Dump data summary reports in a specific format
  - A more readable form, such as changing technical jargon to English
  - Additional annotations
  - Additional time/date/tracking information.
- Eliminate repetitive tasks.

This section explains how to write scan macros using DUMPSCAN subcommands and describes the DVFXEDIT profile. You should be familiar with the REstructured eXtended eXecutor (REXX) language. See the *z/VM: REXX/VM User's Guide* and the *z/VM: REXX/VM Reference* for more information on REXX. You should also be familiar with XEDIT. See the *z/VM: XEDIT User's Guide* for more information on XEDIT.

This section describes how to use some XEDIT subcommands, but it is not intended to give you a complete guide to writing macros. For more examples on writing and using DUMPSCAN macros, refer to the README SAMPLE file which is shipped as part of the Dump Viewing Facility product.

What Is a DUMPSCAN Macro?

A DUMPSCAN macro is a file you invoke from the DUMPSCAN environment. This environment exists whenever DUMPSCAN is being used.

You execute a macro the same way you execute a DUMPSCAN subcommand. Type the macro name on the command line and press Enter or invoke the macro by using a PF-key. See the "SCAN Subcommand" on page 49 for further information on PF-key assignments. A macro can be executed by entering only its name or its name and any parameters needed for its execution.

A macro file can contain:

- DUMPSCAN subcommands
- XEDIT subcommands
- REXX instructions
- REXX or EXEC2 EXECS
- CMS and CP commands.

⚠ **Attention:** A DUMPSCAN macro file must never contain a command or subcommand with the same name as the macro. If it does, the macro will invoke itself recursively. To avoid recursion, change the name of the DUMPSCAN macro file.

Creating a Macro File

A macro is a normal CMS file. It may be created in any of the ways that CMS provides for file creation. Like any CMS file, a macro is identified by file name, file type, and file mode. The macro must conform to these rules:

- File name may be 1 to 8 alphabetic characters in length
- File type must be SCAN, EXEC, or XEDIT
- File mode can be any of the disks to which you have write access, usually your A disk.

Using DUMPSCAN Subcommands in a Macro

A macro can contain any DUMPSCAN subcommand. Most subcommands look for specific data in the dump, format it, and write it to the session record. However, the DUMPSCAN macro subcommands are meaningful only from within a macro, because they pass information to VM/REXX.

What Is an Environment?

When you write a macro, you need to know which command processor is interpreting your command. The interpreter looks at the instructions first within a macro. If the instructions are not XEDIT instructions, they are passed to the environment for interpretation. The environment is the command processor that gets the instructions after VM/REXX has done any symbolic substitution.

You can specify the environment with the REXX instruction ADDRESS:

- ADDRESS SCAN causes the macro to be passed to DUMPSCAN.
- ADDRESS XEDIT causes the macro to be passed to XEDIT.
- ADDRESS CMS causes the macro to be passed to CMS.
- ADDRESS by itself causes the macro to be passed to the default environment.

When you use XEDIT as the file type for your macro, XEDIT is the default environment. When you use EXEC as the file type for your macro, CMS is the default environment. When you use SCAN as the file type for your macro, DUMPSCAN is the default environment.

DUMPSCAN Macro Subcommands

FINDSTRG and READSTRG are subcommands that you can use only from within a macro. READSTRG lets you read from the dump and place that data directly into a REXX variable. FINDSTRG lets you locate data in the dump and put the address of the matching data into a REXX variable. With these two commands, you can extract and format any addressable data within the dump.

NOTE is a subcommand for annotating the session file. It can also be used to annotate the print file produced by DUMPSCAN subcommands when the PRINT ON subcommand is used.

The remaining DUMPSCAN subcommands can be used in a macro, but the formatted output from those subcommands is placed in the session file. If you need this subcommand output, you can do either of the following:

- Use the DVFSTACK macro subcommand to get the formatted output into the program stack. For more information, see "DVFSTACK Subcommand" on page 42.
- Use XEDIT subcommands to get the formatted output into the program stack or into REXX variables.

See "DUMPSCAN Macro Examples" on page 15 for an example of how the output of the CPU subcommand is taken from the session file. Not all DUMPSCAN subcommands are used in the examples, but the technique is the same.

Communicating between the Editor and the Interpreter

The READ and EXTRACT subcommands of XEDIT can supply the macro with data from the DUMPSCAN command line or from the session file. READ takes information from the screen and places it in the program stack. The information in the stack can be the command line, changed lines, prefix area, and program function key definitions (PF keys). The macro gets the information from the program stack with the REXX PULL instruction. EXTRACT can supply a macro with information about internal XEDIT variables or about file data. The information is returned in one or more REXX variables, which can be examined or used by the macro.

READ Subcommand

When a READ subcommand is entered from a macro, the editor redisplays the current screen and waits for the user to press Enter or a PF key. After a key is pressed, the requested data is placed in the program stack.

READ operands can be used to specify how much information is placed in the program stack. A subsequent REXX PULL instruction assigns the data to program variables, and the macro continues.

EXTRACT Subcommand

EXTRACT is issued from a macro. The information is returned in REXX variables. EXTRACT returns information about editor variable settings. These are needed when the macro changes an XEDIT variable and restores it before ending. The example under "DUMPSCAN Macro Examples" uses EXTRACT to get records from the data file and to determine data-file parameters such as the number of records in the file and the position of the current line. EXTRACT has many options. For a complete discussion of the options, read the *z/VM: XEDIT Commands and Macros Reference*.

Displaying Data on the DUMPSCAN Screen

To move prompts and messages from the macro to the DUMPSCAN screen you can use the XEDIT subcommands MSG, EMSG, CMSG, and INPUT.

MSG
Displays a message on the message line.

EMSG
Displays a message on the message line and sounds the alarm.

CMSG
Displays a message on the command line.

INPUT
Puts the message or text in the session file following the current line and makes the new line the current line.

DUMPSCAN Macro Examples

The following examples show you how to use a combination of XEDIT subcommands and DUMPSCAN subcommands to create a new DUMPSCAN function. In Figure 3 on page 15, a new DUMPSCAN subcommand is created by a macro that displays 16 bytes from a specified offset within the prefix page. To get the same information using DUMPSCAN subcommands from the command line would require you to enter the CPU subcommand, write down the prefix page address, add the offset to that address, then enter the DISPLAY subcommand.

```
00001 /* DUMPSCAN Subcommand Macro                    */
00002 /* Display the data at the specified offset from */
00003 /* the failing processor prefix page.            */
00004 /*                                               */
00005 Parse arg offset .
00006 Parse source . . macroname .
00007 'EXTRACT/LINE/MSGMODE'
00008 If offset = '' Then Do
00009   Emsg 'Command format is:' macroname 'offset'
00010   Cmsg macroname
00011   exit
00012   end
00013 'EXTRACT/SIZE/'
00014 Address SCAN 'CPU'
00015 ':'size.1+1 'EXTRACT/CURLINE/'
00016 parse var curline.3 . . . . . . . pfxpgad .
00017 'SET MSGMODE OFF'
00018 ':'size.1+1 'DEL *'
00019 'SET MSGMODE' MSGMODE.1 MSGMODE.2
00020 ':'line.1
00021 Address SCAN 'Display' d2x(x2d(pfxpgad)+x2d(offset)) 'F OFFSET'
00022 exit
```

Figure 3. Example of a DUMPSCAN XEDIT Macro

A description of how the macro works follows:

`00001 /* DUMPSCAN Subcommand Macro */`
`00002 /* Display the data at the specified offset from */`
`00003 /* the failing processor prefix page. */`
`00004 /* */`
An interpreter comment. The first line must be a comment.

`00005 Parse arg offset .`
Take the first operand and assign it to the variable *offset*. The period says to disregard any other operands.

`00006 Parse source . . macroname .`
Get the macro name from CMS. Again, the periods mean that any other file information is not needed.

`00007 'EXTRACT/LINE/MSGMODE'`
Return the line number at the XEDIT current line and the XEDIT message settings.

`00008 If offset = '' Then Do`
Check for the required operand. If it was not entered, issue an error message to the user. DO is the first statement of a series of instructions. This set of statements ends at line 00012.

`00009 Emsg 'Command format is:' macroname 'offset'`
Tell the user the correct way to use this macro.

`00010 Cmsg macroname`
Put the macro name back on the command line in case the user wants to try the macro again.

`00011 exit`
Return control to the caller, because the macro cannot do anything.

`00012 end`
End the DO statement started at line 00008.

`00013 'EXTRACT/SIZE/'`
Use the XEDIT EXTRACT subcommand to find the size of the current file.

`00014 Address SCAN 'CPU'`
Change the environment to SCAN for the following subcommand. SCAN is the name of the DUMPSCAN environment. CPU is a DUMPSCAN subcommand that lists the CPU names and the prefix page address of all CPUs contained in the dump.

`00015 ':'size.1+1 'EXTRACT/CURLINE/'`
Reset the current line to the CPU subcommand first output line and EXTRACT the line into the CURLINE.3 variable.

`00016 parse var curline.3 . . . . . . . pfxpgad .`
Break the curline variable into its parts and, ignoring some parts, assign the prefix page address to the *pfxpgad* variable.

`00017 'SET MSGMODE OFF'`
Use XEDIT SET to suppress any messages.

`00018 ':'size.1+1 'DEL *'`
Remove the CPU subcommand output from the file.

`00019 'SET MSGMODE' MSGMODE.1 MSGMODE.2`
Turn the messages back to their previous settings.

`00020 ':'line.1`
Reset the current line to its position when you started.

`00021 Address SCAN ‘Display’ d2x(x2d(pfxpgad)+x2d(offset)) ‘F OFFSET’`
Switch to the DUMPSCAN environment, and after the interpreter works through the calculation, enter the DUMPSCAN DISPLAY subcommand with the OFFSET operand for 15 bytes.

`00022 exit`
Indicate that the macro is finished.

Figure 4 on page 17 shows a macro that does the same thing as the one in Figure 3 on page 15, but in a SCAN environment.

```
/* DUMPSCAN Subcommand Macro                  */
/* Display the data at the specified offset from */
/* the failing processor prefix page.           */
/*                                              */
Parse arg offset .
Parse source . . macroname .
If offset = '' Then Do
  Address 'XEDIT'
  Emsg 'Command format is:' macroname 'offset'
  Cmsg macroname
  exit
  end
Address SCAN 'DVFSTACK ON'
Address SCAN 'CPU'
pull . . . . . . . pfxpgad .
Address SCAN 'DVFSTACK OFF'
Address SCAN 'Display' d2x(x2d(pfxpgad)+x2d(offset)) 'F OFFSET'
exit
```

Figure 4. Example of a DUMPSCAN SCAN Macro

DVFXEDIT Profile

DVFXEDIT XEDIT is a macro available with the Dump Viewing Facility that modifies a standard XEDIT session to the special DUMPSCAN format. This macro can be modified to change the session format to your preference. You should not change the definition of ENTER.

Assigning Program Function Keys to DUMPSCAN Subcommands

The PF keys default to XEDIT functions in the DUMPSCAN environment, and are assumed to be for XEDIT functions. You can assign keys to DUMPSCAN subcommands or DUMPSCAN subcommand macros by putting an XEDIT SET command in a copy of the DVFXEDIT XEDIT macro that you keep on your A-disk.

For example, you may want to assign PF keys 4 and 5 to the DUMPSCAN FORWARD and BACKWARD subcommands. You would put the XEDIT subcommands SET PF04 SCAN BACKWARD and SET PF05 SCAN FORWARD in your DVFXEDIT XEDIT macro. SET is an XEDIT subcommand name that changes XEDIT system variables. PF04 is the name of an XEDIT system variable. SCAN is the CMS command that gives you access to DUMPSCAN subcommand processing. BACKWARD is the name of the DUMPSCAN subcommand that displays addresses lower than the last addresses displayed.

Scenario 1: Analyzing a CMS Program Exception

In this scenario, CMS has terminated abnormally with an operation exception at address 80000002. The steps that follow suggest one way that this problem could be analyzed.

Step 1: Checking the Error Messages

The user was notified of the problem by the messages:

```
    DMSITP143T  Operation exception occurred at 80000002 in
                system routine EXEC; re-IPL CMS
    DMSABE2047I AUTODUMP dump started; please wait
    DMSABE1297I Dump has been taken
    DMSDIE3550I All APPC/VM and IUCV paths have been severed.
```

DMS in the messages indicates that the messages are from CMS. The **ITP, ABE,** and **DIE** indicate that DMSITP, DMSABE, and DMSDIE issued the messages.

The message numbers are **143T, 2047I, 1297I,** and **3550I**. For more information on these messages, see the *z/VM: Other Components Messages and Codes*. Also, online help can be used by issuing:

```
HELP DMSITP143T  or  HELP DMS143T  or  HELP MSG DMS143T
```

Note: Always start with the first message because this is usually the best indication of the problem.

Step 2: Use DUMPLOAD to Process the Dump

Re-IPL CMS and order the reader before issuing the DUMPLOAD command. In the following output, the x is the version and y.z is the release of your z/VM system.

```
dumpload
HCPEDZ8183I DUMPLOAD z/VM VERSION x RELEASE y.z
HCPEDZ8150I PROCESSING z/VM DUMP PRB00000 DUMP0001 A
HCPEDZ8167I VIRTUAL MACHINE DUMP FROM z/VM V0xR0yMz
HCPEDZ8168I VIRTUAL MACHINE DUMP, FORMAT=CMS,
DUMPID=
HCPEDZ8156A DO YOU WANT TO PROCESS THIS DUMP? (YES/NO)
yes
```

Step 3: Use DUMPSCAN to Analyze the Dump

The format of the DUMPSCAN command is:

```
dumpscan prb00000

HCSDSS200I PROCESSING FILE PRB00000 DUMP0001 A1     09/22/00    16:56:00
HCSDSS401I READY, DUMP TYPE IS VM
```

The DVF session that follows will be contained in an XEDIT file called PRB00000 VIEW0000 A1. This can be saved at the end of the DVF session by issuing FILE. Because DVF uses XEDIT, the function keys F7 and F8 can be used to scroll backward and forward in the file.

CMS Data Areas and Control Blocks contains the mapping for NUCON (CMS's Nucleus Constant Area) and PGMSECT (CMS's Program Interrupt Work Area). NUCON is always at address 0 in a CMS dump. The address of PGMSECT is pointed to by APGMSECT in NUCON, which is at X'654'. The first DVF command entered is REGS:

```
----> regs
REGS
CPU ADDRESS - 0000                        PREFIX REGISTER - 00000000
GENERAL REGS  0 - 15
 00001000 000081F4 0000001B 00008070  00F53BB8 0000820F 00000001 0000000E
 00000000 00EF0000 00F53960 00EE6748  80F075D2 00EFD268 80F53AD2 00000000
CONTROL REGS  0 - 15
 000110E2 00000000 00000000 00000000  00000000 00000000 FF000000 00000000
 00000000 00000000 00000000 00000000  00000000 00000000 C2000000 00000000
ACCESS  REGS  0 - 15
 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
FLOATING POINT REGS  0 - 6
 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000

TOD CLOCK         B4AF5ED0 31C3E701              PSW 00080000 80F53AE2
CLOCK COMPARATOR  00000000 00000000
CPU TIMER         FFFFFE1D 034F2419

EXT OLD 030C1000 8107D94C INT CODE 0080      EXT NEW 00080000 80F2   2D88
SVC OLD 00081000 80F19E90 INT CODE 00CC ILC 0002 SVC NEW 000C0000 80F1120E
PGM OLD 000C0000 80000002 INT CODE 0001 ILC 0002 PGM NEW 00080000 813A99F8
MCH OLD 000C2000 80F12130                        MCH NEW 00080000 80F3E390
I/O OLD FFFFFFFF FFFFFFFF                        I/O NEW 00080000 80F10910
```

According to this dump, the program old PSW (PGM OLD) is 000C0000 80000002 and the INT CODE is 0001, which indicates an operation exception occurred at 0.

Next, the DISPLAY command, (minimum abbreviation is d), is used to look at offset X'654' to determine the address of PGMSECT:

```
----> d 654.20
DISPLAY 654
  00000650  00F24B34  000078C8  00001378  00F14180 F6 *.2.....H.....1..*
  00000660  00004F78  00F51A42  0000AB18  00F142DC    *..|..5.......1..*
  00000670  00F0F44C  00001000  00F152E4  00F1DA00    *.04<.....1.U.1..*
```

To obtain only one line of output, enter:

```
----> d 654.08
DISPLAY 654
  00000650  00F24B34  000078C8  00001378  00F14180 F6 *.2.....H.....1..*
```

The address of PGMSECT is at 78C8. CMS Data Areas and Control Blocks contains the mapping for PGMSECT (CMS's Program Interrupt Work Area). Of particular interest is offset 60 which contains the instruction length and interrupt code and beginning at offset 7C in PGMSECT is PSAVE, which contains the registers at the time of the program interrupt. To look at the offsets for this control block, it is convenient to use the DVF Display command with the OFFSET option. Since PGMSECT starts at 78C8, use the DVF command D 78C8 OFFSET to display the offsets.

```
----> d 78c8 offset
DISPLAY 78C8                       OFFSET
      0000  00000000  00000000  00000000  00000000 F6 *................*
      0010  C5D7C9C5  00000000  00001000  01069FDD    *EPIE............*
      0020  00000000  00000001  00000000  00EFD104    *..............J.*
      0030  00EFD420  00EE3D70  00003AC0  0106AB0C    *..M.............*
      0040  01069B0D  00000000  81068B0E  00EFD268    *........a.....K.*
      0050  8106A00A  00000000  000C0000  80000002    *a...............*
      0060  00020001  00000000  00000000  00000000    *................*
      0070  00000000  00000000  00000000  00001000    *................*
      0080  01069FDD  00000000  00000001  00000000    *................*
      0090  00EFD104  00EFD420  00EE3D70  00003AC0    *..J...M.........*
      00A0  0106AB0C  01069B0D  00000000  81068B0E    *............a...*
      00B0  00EFD268  8106A00A  00000000  00000000    *..K.a...........*
      00C0  00000000  00000000  00000000  00000000    *................*
      00D0  00000000  00000000  00000000  00000000    *................*
      00E0  00000000  00000000  00000000  00000000    *................*
      00F0  00000000  00000000  00000000  00000000    *................*
      0100  00000000  00000000  00000000  00000000    *................*
      0110  00000000  00000000  00000000  00000000    *................*
      0120  40404040  40404040  40404040  40404040    *                *
      0130  40404040  40404040  00000000  00000000    *        ........*
      0140  00000000  00000000  00000000  00000000    *................*
```

Offset 60 in PGMSECT is 00020001: ILC 0002 and INT CODE 0001. Beginning at offset 7C in PGMSECT are registers 0-15. Because this operation exception occurred at low core 0, it is suspicious that register 15 contains a 0. CMS most often uses a BALR 14,15. If Register 15 contains a zero, then CMS probably branched and linked there, and Register 14 will probably contain the return address of the BALR, in this case, 8106A00A.

```
----> d 106a00a.40
DISPLAY 106A000
  0106A000  41110001  58F00328  05EF12FF  4780A51B F4 *.....0........v.*
  0106A010  9110D22E  4780A513  41600006  45E09264    *j.K...v..-....k.*
  0106A020  41600019  47F0A849  5010D200  BF6FD1BC    *.-...0y.&;K..?J.*
  0106A030  4780A61D  95E7D22C  4780A61D  5010D350    *..w.nXK...w...L.*
```

At address 106A008 is 05EF (BALR 14,15).

The previous instruction at 106A004 is 58F00328 (L instruction of register 15 with the contents at X'328'). From NUCON, low core 328 is NUCAFROC that is the address of DMSFROBC, which should contain an address in the CMS nucleus. Displaying 328 shows that this contains 0.

```
----> d 328.30
DISPLAY 328
  00000320  00000000  00000000  00000000  00F1406C F6 *.............1 %*
  00000330  00F14078  00F14084  00000000  00000000    *.1 ..1 d........*
  00000340  0000B3A0  00009170  00000000  00003C60    *......j........-*
  00000350  00003C60  00000000  00000000  00000000    *...-............*
```

Step 4: Summarizing the DUMP Analysis

The operation exception occurred when a BALR to storage management module DMSFROBC failed because the address in NUCON is zero. There are two possible causes for this problem. One possibility is that the CMS nucleus was built incorrectly. If IPL CMS followed by D 328 does not show the correct nucleus address of DMSFROBC, then the systems programmer needs to rebuild CMS properly. The more likely possibility is that DMSFROBC was overlayed. The dump does not indicate how NUCON 328 became zero, but because DMSFROBC is a nucleus constant, the CP trace command:

```
CP TRACE STORE INTO 328
```

would indicate how the overlay occurred.

Chapter 3. Command Reference

In this chapter, the Dump Viewing Facility commands are described in alphabetic order. The description of each command includes format, operands, return codes, options, and responses, if any. Where applicable, usage notes further describe the characteristics of the command. For more information on messages, see *z/VM: Other Components Messages and Codes*. You enter Dump Viewing Facility commands from a terminal attached to a CMS virtual machine.

Using the Online HELP Facility

You can receive online information about the commands described in this document using the z/VM HELP Facility. For example, to display a menu of DUMPVIEW commands, enter:

```
help dumpview menu
```

To display information about a specific DUMPVIEW command (ADDMAP in this example), enter:

```
help dumpview addmap
```

You can also display information about a message by entering one of the following commands:

```
help msgid or help msg msgid
```

For example, to display information about message HCSDSS200I, you can enter one of the following commands:

```
help hcsdss200i or help hcs200i or help msg hcs200i
```

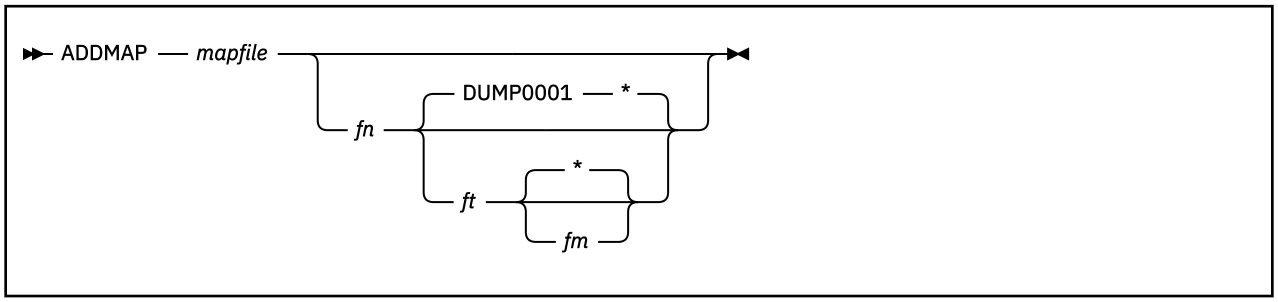
For more information about using the HELP Facility, see the *z/VM: CMS User's Guide*. To display the main HELP Task Menu, enter:

```
help
```

For more information about the HELP command, see the *z/VM: CMS Commands and Utilities Reference* or enter:

```
help cms help
```

ADDMAP Command



Purpose

Use the ADDMAP command to append virtual machine module maps.

Operands

mapfile
is the file name, file type, and file mode of the input CMS file containing the module map.

fn
is the file name of a CMS file containing the dump to which you want the module map appended.

ft
is the file type of a CMS file containing the dump to which you want the module map appended. The default is DUMP0001.

*****
fm
is the file mode of the CMS file containing the dump to which the module map will be appended. The * is the default.

Usage Notes

1. If you add an incorrect map to a dump, you can start ADDMAP again to add the correct map to the dump.
2. In order to use the ADDMAP command, the load map must first have been processed by the MAP command. For more information, see the "MAP Command" on page 26.
3. If you do not specify a dump file name, ADDMAP prompts you for one. If you do not specify the file mode, or if you specify *, the system uses the standard CMS search sequence.

Examples

1. You have a processed module map with the default name for a CMS dump (CMSDVF MAP A1). To resolve the module map and append it to dump DUMPUSER DUMP0001 A1, enter:

```
addmap cmsdvf map a1 DUMPUSER dump0001 a1
```

   For more information on default map names, see the MAP command.

Messages and Return Codes

Return Code
Explanation

0
Successful completion

20
: Invalid file ID

24
: Command line error

28
: Nonexistent CMS file

32
: Invalid data in file

36
: Disk not accessed

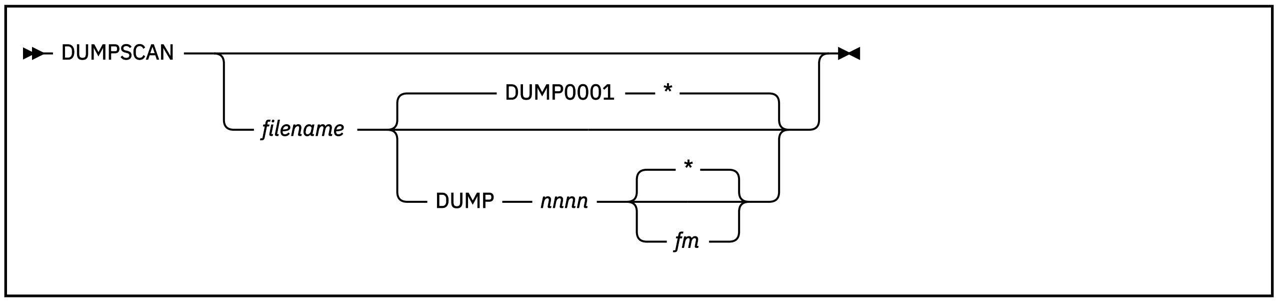
41
: Insufficient storage

50
: CP dumps are not supported. Use VM Dump Tool

100
: FSREAD/PRINTL error

104
: Internal processing error

DUMPSCAN Command



Purpose

Use DUMPSCAN for interactively analyzing and debugging problems in a dump. After you start a session, you can use the DUMPSCAN subcommands described in Chapter 5, "DUMPSCAN Subcommand Reference," on page 51.

Operands

filename
is the file name of the CMS file containing a dump to be processed.

DUMP*nnnn*
is the file type of the dump to be processed. The file type must be eight characters long, consisting of the string `DUMP` and a four-digit number, *nnnn*. The default file type is `DUMP0001`.

*****
fm
is the file mode of the dump file. If you do not specify the file mode, or if you specify an asterisk, DUMPSCAN uses the standard CMS search sequence.

Usage Notes

1. The DUMPSCAN session file has the same file name as the dump file. The file type is `VIEW`*nnnn* where *nnnn* is the number from the dump file type, DUMP*nnnn*.
2. DUMPSCAN modifies and re-records the symptom record to contain the dump name. Any modification made by the guest through user exits is also recorded.
3. If the dump is in one of the supported formats and the corresponding extraction routine is available, DUMPSCAN runs the routine to update information in the dump's symptom record and information record. See Appendix A, "Using Attachment Interfaces," on page 139 for information about exit routines and supported dump types.
4. If you enter DUMPSCAN without operands, you are prompted for the dump file name, file type, and file mode.
5. The dump viewing session can be filed on your A-disk with the XEDIT FILE subcommand. When you view the same dump later, the saved file is reactivated and the new session is appended to the *dumpname* DUMP*nnnn* file containing the previous sessions. See the usage guide section under "Viewing Several Dump Files at a Time" on page 11 as well as the description of the DUMPSCAN subcommand in Chapter 5, "DUMPSCAN Subcommand Reference," on page 51 for an explanation on how to view multiple dump files from the DUMPSCAN command environment.

Messages/Return Codes

Return Code
Explanation

0
: Successful completion

20
: Invalid file ID

24
: Command line error

28
: Nonexistent CMS file

32
: Invalid data in file

36
: Disk not accessed

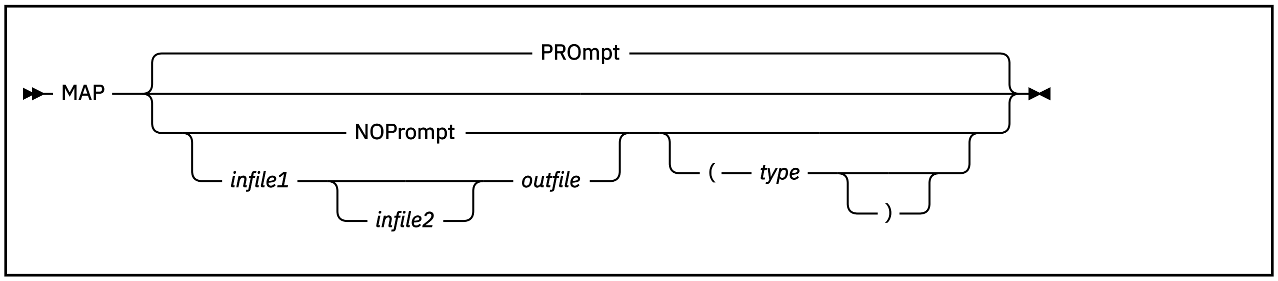
41
: Insufficient storage

50
: CP dumps are not supported. Use VM Dump Tool

100
: FSREAD/PRINTL error

104
: Internal processing error

MAP Command



Purpose

Use the MAP command to convert a load map into a format that the Dump Viewing Facility can process. The converted load map, which is called the **module map**, serves as input to the ADDMAP command. ADDMAP appends the converted load map to the dump file. The Dump Viewing Facility requires primary and secondary load maps to create certain types of module maps.

When you invoke MAP, you may request or suppress prompting. You may specify the files or accept the default files for processing.

Operands

PROmpt
requests prompting. PROMPT is the default. Prompting occurs in the following sequence:

1. The Dump Viewing Facility asks first for one of the following:

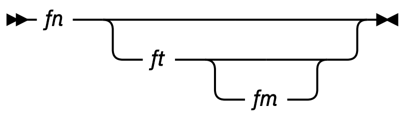
   type
   specifies the type of module map to create. See Table 6 on page 27 for a list of types.

   null line
   accepts the default type: CMS.

   SUBSET
   invokes CMS subset mode.

   HX
   terminates the MAP command.

2. You may then supply output and input files:



specifies each file to process. You are prompted for the output file ID, the primary input file ID, and the secondary input file ID (if necessary). If you do not specify the file type or file mode, the file type defaults to MAP, and the file mode defaults to *.

null line
accepts a predefined file ID. See Table 6 on page 27 for default input and output map file IDs.

SUBSET
invokes CMS subset mode.

HX
terminates the MAP command.

NOPrompt
suppresses prompting altogether. The Dump Viewing Facility uses the default input and output file IDs for the module map type that you specify. Table 5 shows the file IDs associated with each type.

Note: SFS includes coordinated resource recovery facility.

infile1
indicates the file name, file type, and file mode of the input CMS file containing the primary load map.

infile2
indicates the file name, file type, and file mode of the input CMS file containing the secondary load map required to create certain types of module maps. To create a module map for a supported type that requires a secondary load map, MAP processes the CMS nucleus load map (*infile1*) and then the secondary load map (*infile2*).

outfile
is the file name, file type, and file mode of the resulting CMS file containing the module map, which is used as input to the ADDMAP command.

Options

type
specifies a type of module map for the MAP command to create. You may specify one of the module map types listed below. If you do not specify a module map type, *type* defaults to CMS.

The following table lists the module maps which may be created using the Dump Viewing Facility. If you wish to create another type of module map see, Appendix A, "Using Attachment Interfaces," on page 139.

Table 6. Maps Processed by the MAP Command

Module Map Type	Module Map Component	Load Map Type	Nucleus	Default Input Map File	Default Output Map File
CMS	Primary	CMS	Yes	CMSNUC MAP *	CMSDVF MAP A
DFSMS	Primary	CMS	Yes	CMSNUC MAP *	DFSMSDVF MAP A
	Secondary	DFSMS	No	FSMDFSMS MAP *	
GCS	Primary	GCS	Yes	GCSNUC MAP *	GCSDVF MAP A
PVM	Primary	CMS	Yes	CMSNUC MAP *	PVMDVF MAP A
	Secondary	PVM	No	PVM MAP *	
SECP	Primary	GCS	Yes	GCSNUC MAP *	SECPDVF MAP A
	Secondary	SECP	No	SECP MAP *	
TSAF	Primary	CMS	Yes	CMSNUC MAP *	TSAFDVF MAP A
	Secondary	TSAF	No	TSAF MAP *	

Usage Notes

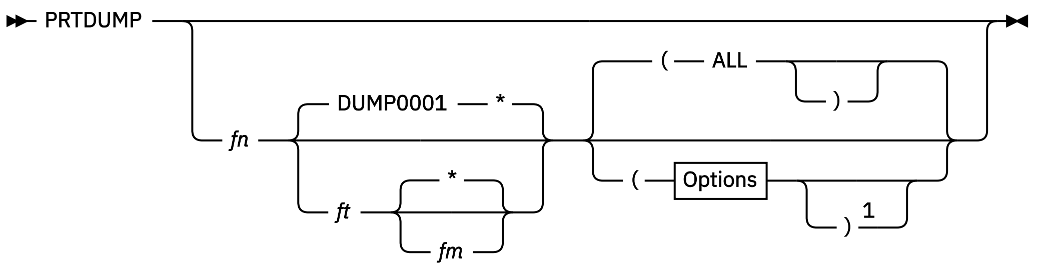
1. MAP requires primary and secondary load maps for certain module map types (see Table 6 on page 27). If you specify a file ID for one of these types on the command line, you must specify *infile2*. If you do not specify *infile2*, you receive an error message and command processing terminates.

   Other map types do not require secondary maps. If you specify *infile2* for one of these types, you receive an error message and command processing terminates.
2. MAP does not issue prompting messages when you specify file IDs or enter NOPROMPT on the command line. If an error occurs or you do not specify a required file ID, you receive an error message and command processing terminates.
3. AVS, RSCS, and SFS do not have separate maps. AVS and RSCS require that the GCS NUCLEUS map be added to their dumps. SFS requires a CMSNUC map be added to the dump.

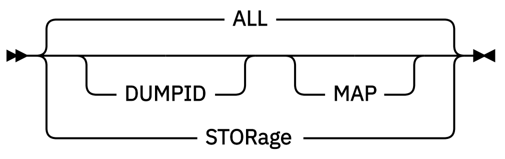
Messages and Return Codes

Return Code	Explanation
0	Successful completion
20	Invalid file ID
24	Command line error
28	Missing files or file already exists
32	Invalid data in file
36	Disk not accessed
41	Insufficient storage
50	CP dumps are not supported. Use VM Dump Tool
100	FSREAD/PRINTL failure
104	Internal processing error

PRTDUMP Command



Options



Notes:

1 You can enter Options in any order between the parentheses.

Purpose

Use the PRTDUMP command to print summary information about the major system control blocks and data areas of the dumped system.

Use the summary reports to assist in problem analysis. Each summary report is designed as a reference to information in the dump. You refer to it when interactively viewing a dump using the DUMPSCAN command.

The printed summary information for control blocks includes key fields rather than the entire block. The data and flags are interpreted, and a text description is provided wherever possible. Only control block data that is used most often for debugging is presented. The formats for the printed dumps are described in "Printed Dump Format" on page 31.

Using the Dump Viewing Facility PRTDUMP command, you can print these summary reports:

- Symptom Records

  The dump symptom record contains information that indicates the state of the system when the dump is taken. The printed information is in the same format as output obtained through the SYMPTOM subcommand of DUMPSCAN.

- General Processor Information

  This report contains information that describes the processors associated with the dump. This report includes:

  - Registers
  - Clocks and timer
  - CPU address and prefix register
  - Program status words (PSWs).

  For 370-mode virtual machine dumps, the general processor summary report also includes:

  - Channel status word (CSW)
  - Channel address word (CAW)

  - Interval timer
  - Current PSW.
- Dump Viewing Facility Module Map

  Module and entry point names and their addresses in the dump are printed for all modules in real storage when the dump is taken.
- DUMPID
  - Valid only for virtual machine dumps.
- Virtual Machine Dump Specific Summary Reports

Operands

fn
: is the file name of the CMS file containing the dump to be processed. It is a 1- to 8-character string containing any combination of the characters 0–9, A–Z, @, #, -, _, +, :, and $.

ft
: is the file type of the dump file. The default is DUMP0001.

fm
*****
: is the file mode of the CMS dump file. If the file mode is not specified or an asterisk (*) is specified, PRTDUMP uses the standard CMS search sequence.

Options

(
: indicates that PRTDUMP options follow. If the (is specified with no options following, ALL is the default. If (is not specified and options follow the file mode, they are treated as invalid options. If nothing follows the file mode, ALL is the default.

STORage
: causes all the storage in the dump to be printed. If the CP option is specified, only the control program pages in the dump are printed.

ALL
: specifies that all applicable reports should be printed for the dump type.

  Specifying ALL does not cause storage pages to be printed. You must specify the STORAGE option on a separate invocation of PRTDUMP in order to print dumped storage.

  For virtual machine dumps, ALL specifies that symptom record data, general processor information, DUMPID, and the module map be printed. ALL is equivalent to specifying MAP and DUMPID on the command line. If no operands are specified, ALL is the default.

DUMPID
: specifies that the identifier of the virtual dump should be printed. This option applies only to virtual machine dumps.

MAP
: specifies that the Dump Viewing Facility module map that is appended to the dump should be printed. However, the dump file must be on a CMS disk to which you have write access.

Usage Notes

1. If you enter the PRTDUMP command without any operands, you are prompted to enter the dump file name and the file mode, or HX to end processing. After you respond to the prompt, the summary reports is printed.

   This is equivalent to entering the ALL option, which is the default.

The STORAGE option must be used separately from the ALL option. In order to print storage from the dump, specify the STORAGE option on a separate invocation of PRTDUMP. If the STORAGE option was used to print the storage from the dump, and if the format or type of virtual machine dump had previously been set either by the CP VMDUMP command or by a previous invocation of the DUMPSCAN FORMAT subcommand, then the dump type must be changed to the default value "FILE" by re-issuing the FORMAT subcommand of DUMPSCAN prior to issuing PRTDUMP. To return to the original dump type environment, enter the FORMAT subcommand again and specify the desired virtual machine dump type.

2. If the file mode is not specified or an asterisk (*) is specified, the system uses the standard CMS search sequence.
3. The symptom record summary and general-processor information summary reports are printed for every valid invocation of the PRTDUMP command. They are the first reports to be printed.
4. For virtual machine dumps, PRTDUMP formats the crypto domain index register for ESA virtual machines that have the Integrated Cryptographic Facility defined.
5. If the dump symptom record is missing or not readable, an error message is issued, and you are prompted to continue dump processing.

Messages and Return Codes

Return Code	Explanation
0	Successful completion
20	Invalid file ID
24	Command line error
28	CMS file does not exist
32	Invalid data in file
36	Disk not accessed
41	Insufficient storage
50	CP dumps are not supported. Use VM Dump Tool
100	FSREAD/PRINTL error
104	Internal processing error

Printed Dump Format

The first page of a printed virtual machine dump contains the format and dump ID of the dump (from the VMDUMP command line). The following information for the virtual processor on which the VMDUMP command was issued is also printed on the first page of the dump:

- CPU address
- General-purpose registers
- Control registers
- Floating-point registers

- Access registers (if the virtual machine was an XA, ESA or XC virtual machine)
- TOD clock
- TOD clock comparator
- CPU timer
- Prefix register (for the virtual machine)
- Program status word (PSW)
- Crypto Domain Index Register (if the mode of the virtual machine is ESA and a virtual ICRF was defined).

The prefix page of the virtual processor on which the VMDUMP command was issued is printed, followed by the remainder of storage pages. Each line of the printed dump contains the following:

- Guest real address of the data
- Eight fullwords of hexadecimal data
- EBCDIC translation of the hexadecimal data
- Storage key (if the guest real address lies on a 4 KB boundary).

Note: Access registers are dumped only if the virtual machine is an XA, ESA or XC virtual machine.

Because the prefix page is the first storage page printed, when it is encountered a second time, the following line is printed:

```
SKIPPING PREFIX AREA
```

Storage pages that are not included in the dump are indicated by the line:

```
nn TO mm SUPPRESSED NON-CONTROL-PROGRAM PAGE(S)
```

where *nn* and *mm* are the addresses of the pages not included in the dump.

Duplicate lines are suppressed and are indicated by the line:

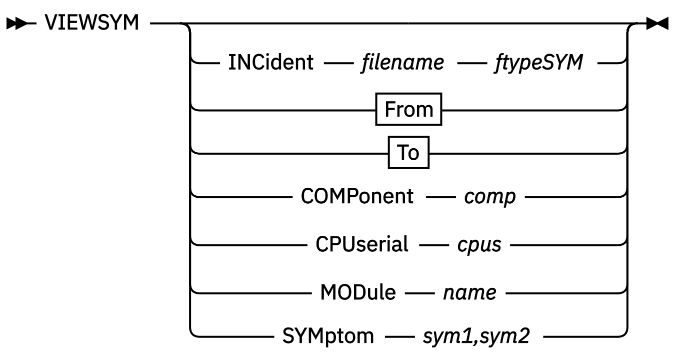
```
nn TO mm SUPPRESSED LINE(S) SAME AS ABOVE
```

where *nn* and *mm* are the addresses of the data that is not included in the dump.

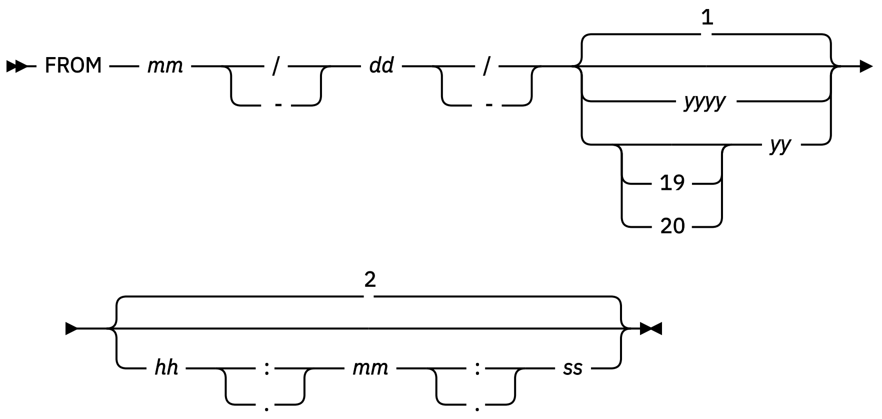
Page ejects are performed when necessary, and when all of storage has been dumped, the printed dump ends with the line:

```
*** END OF DUMP ***
```

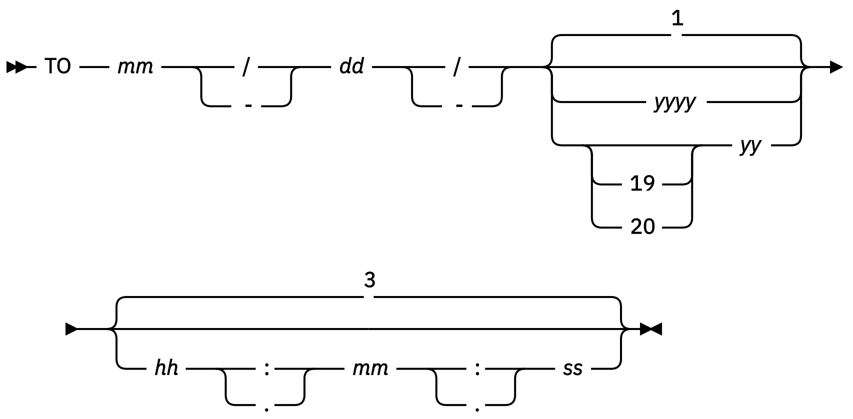
VIEWSYM Command



From



To



Notes:

[1] The default is the earliest found.
[2] The default is the beginning of the day.
[3] The default is the end of the day.

Purpose

Use the VIEWSYM command to select and view CMS symptom record files created by the RETRIEVE SYMPTOM command. These are located on the A-disk of the virtual machine that issued the RETRIEVE

SYMPTOM command. The symptom record files may then be erased to recover space on the symptom record recording virtual machine's A-disk.

Symptom records can be selected for viewing through a number of criteria such as over a period of time, by processor ID, by incident number, or by symptom string. A collection of symptom records satisfying the specified criteria is placed in a separate CMS file as a result of this command.

Operands

(blank - no entry)
indicates that no operands are specified. When this happens, a command menu is displayed. You can then specify selection criteria from this menu. See VIEWSYM Command Menu for a description of the command menu.

INCident *filename ftype*SYM
is the file name and file type of the desired symptom record. When INCIDENT is specified, the VIEWSYM subcommand will search for the specific symptom record and display it if found. The INCIDENT keyword and the other VIEWSYM keywords are mutually exclusive.

Since the last three characters of the file type are always SYM, they do not have to be specified and can be allowed to default.

FROM *date time*
indicates that symptom records that have a date falling on or after this date and time are selected for viewing. The date is specified as mm/dd/yy; if not specified it defaults to the earliest symptom record found.

Either (/) or (-) can be used to separate the months, days, and years. Leading zeros are not required but can be specified. The year can be entered as either a four-digit or two-digit number or allowed to default to the earliest found. For example, 01/08/87, 1/8/87, 1/8-87, and 1-08/1987 are all valid. A two-digit year will be interpreted within a window of 100 years centered on the current date.

For example, if the year is 1997, the 100-year window spans from 1947 to 2046. Any two-digit year in the range 47-99 will be interpreted as 1947-1999, and any two-digit year in the range 00-46 will be interpreted as 2000-2046. Likewise, if the year is 2003, the 100-year window spans from 1953-2052. Any two-digit year in the range 53-99 will be interpreted as 1953-1999, and any two-digit year in the range 00-52 will be interpreted as 2000-2052.

The time is specified as hh.mm.ss; if not specified, it defaults to start when the day begins. Various combinations are allowed. Either (:) or (.) can be used to separate the hours, minutes, and seconds. If 24:00:00 is specified it is assumed that you mean the next day. Hours, minutes, or seconds need not be specified; they can be allowed to default.

TO *date time*
indicates that symptom records that have a date falling on or before this date and time are selected for viewing. If the date is not specified, it defaults to the latest symptom record found. If the time is not specified, it defaults to the end of the day. If 24:00:00 is specified, it is assumed to mean the beginning of the day. The TO date and time must be later than the FROM date and time.

COMPonent *comp*
is the 9-character component ID not counting any hyphens or blanks.

CPUserial *cpus*
is the six-digit CPU serial number for the selected CPU.

MODule *name*
is a string, from 1 to 10 characters in length, which is the name identifying the module associated with the failure.

SYMptom *sym1,sym2*
is a string, from 1 to 15 characters in length, which contains a symptom string.

For example: AB/SPRG001

Usage Notes

1. The maximum size of the input line—132 characters—determines the number of symptoms that can be searched for at one time (with 10 strings being the maximum).
2. If MODule is specified, the VIEWSYM command will look for the specified value in any RIDS/nnnnnnnnnn symptom string in Section 3.
3. Symptom strings are from 1 to 15 characters in length and contain a Structured Database identifier (SDB ID) followed by a symptom string. The SDB ID is separated from the symptom string by a /. The component and module can be specified as SYMPTOMS by using the SDB ID. The SDB ID for component is PIDS and the SDB ID for module is RIDS.

   For more information on symptom strings and symptom records, see *z/VM: System Operation*.
4. Searching for symptom records uses the following procedure:
   a. Selected symptom records must have a TOD value within the date and time range specified.
   b. Selected symptom records must have the same CPU number if the CPU number is specified.
   c. Selected symptom records must have all the primary symptom strings specified but may contain additional strings. The primary symptom strings may appear in any order.

The VIEWSYM command passes the following return codes to CMS in register 15.

Return Code
Explanation

0
Successful completion

8
Error

VIEWSYM Command Menu: If you do not enter any operands with the VIEWSYM command, the following command menu will be displayed. You can then fill in the selection criteria and press ENTER to invoke the command.

```
              Symptom Viewing Facility - Command Menu
              ---------------------------------------

Enter INCIDENT ____    __SYM to look at a specific incident

              OR any of the following search arguments:

                              DATE                 TIME

       FROM    . . . . .  mm / dd / 19yy        hh : mm : ss
       TO      . . . . .  mm / dd / 19yy        hh : mm : ss

       CPU SERIAL  . . .  nnnnnn

       COMPONENT ID  . .  ___________
       MODULE  . . . . .  ___________
     SYMPTOM   _______________   SYMPYOM   __________
     SYMPTOM   _______________   SYMPYOM   __________
     SYMPTOM   _______________   SYMPYOM   __________
     SYMPTOM   _______________   SYMPYOM   __________
     SYMPTOM   _______________   SYMPYOM   __________
```

Figure 5. VIEWSYM Command Menu

Command Menu Usage Notes:

1. If an incident *filename ftype*SYM is entered along with any other selection criteria, an error message is displayed on the message line.
2. If any of the fields in the command menu are filled in, they are used as selection criteria.
3. If both fields in the command menu are filled in and the command line has data other than QUIT or END, a search will be done and the command line data will be ignored.

4. If you enter QUIT or END, the command terminates and control is passed back to CMS.

List of Matches: An example of an incident list is shown in Figure 6 on page 36. If more than one match is found as a result of a specified search operand, a list of these matches is created in a temporary in-storage file.

```
                  Symptom Viewing Facility - List          10 Matches

INC : FBD00378 BAB44SYM  COMP: 566419601  RIDS/HCPEND     RIDS/HCPEOD
DATE: 12/12/87 14:01:19  CPU : 23456      AB/SPRG0001     REGS/0E248

INC : FBD04276 FDB56SYM  COMP: 566419601  RIDS/HCPBAD     AB/SPRG0004
DATE: 12/13/87 14:05:23  CPU : 23456      REGS/02002      REGS/0311C

INC : FBD04D66 3AD46SYM  COMP: 566419601  RIDS/HCPEND     RIDS/HCPEOD
DATE: 12/14/87 13:59:18  CPU : 23456      AB/SPRG0001     REGS/0E248

INC : FBD05378 B3366SYM  COMP: 566419601  RIDS/HCPDED     AB/SPRG0002
DATE: 12/15/87 14:03:22  CPU : 23456

INC : FBD06422 BA324SYM  COMP: 566419601  RIDS/HCPXXX     RIDS/HCPEOF
DATE: 12/16/87 14:03:35  CPU : 23456      AB/SPRG0006
```

Figure 6. VIEWSYM List Display

List Display Usage Notes:

1. **Primary Symptom Strings -** These symptoms are called primary because they are intended to contain a set of symptom database keywords that will be most valuable in uniquely identifying the failure or event.
2. **Secondary Symptom Strings -** These symptoms are intended to allow a function to store more characteristics of the event without modifying the primary symptom strings.
3. You have the ability to select any one of the matches by either placing the cursor on any line for that record and using the Enter key, or entering the incident *fname ftype* on the command line. If you enter an incorrect incident *filename ftype*SYM on the command line, a message is displayed.
4. If more matches are found than can fit on one screen, you can scroll forward and backward to see other matches.
5. If you press Enter and the cursor is between matches on the screen, a message is displayed informing you how to select a match.
6. If the cursor is on a valid match and the command line or PF key has data other than QUIT or END, the appropriate match is displayed and the data on the command line is ignored.

Individual Symptom Record Display: If you specify an individual symptom record or there is only one match for the search operand, the record itself will be displayed. An example of an individual symptom record is shown in Figure 7 on page 36.

```
              SYMPTOM RECORD FOR INCIDENT  B410EB3F 4832ESYM

TOD CLOCK . . B410EB3F4832EF02                   DATE. . . . . 05/19/00
TIME ZONE . . -04:00:00                          TIME. . . . . 20:15:48

CPU MODEL . . 9672                               BASE SCP. . . 5654
CPU SERIAL. . 026452                             NODEID. . . . GDLVMK4

SPOOLID: 7785                                    DUMP TYPE . . VMDUMP
-----------------------------------------------------------------------
SECTION 5 DATA:
              USERID DUMPED: NAMESRV
              DUMP RECEIVER: NAMESRV
              SPOOLID: 7785
```

Figure 7. VIEWSYM Individual Item Display

Individual Symptom Record Display Usage Notes:

1. In the same manner that it presents a match list, VIEWSYM will invoke XEDIT and create a temporary file, VIEWSYM SESSION, in storage. It will then place the formatted output in that file.
2. If the output spans more than a screen's worth of data, XEDIT provides the scrolling function to enable you to view the entire record.
3. The output consists of the header section, followed by the symptom strings in sections 3 and 4. If you provided any preformatted data in section 5, it will also be displayed.
4. For VM dumps only, an additional line of output is displayed for the Section 5 Data: the number of address spaces, including the primary, that were dumped by the CP VMDUMP command or by DIAGNOSE code X'94'. For CP dump types, this additional line of output is not displayed, because only a single CMS dump file is created by DUMPLOAD.

   The actual number of accessible related CMS dump files may be less than the total displayed if DUMPLOAD processing failed while building the CMS dump files, or if any of the related dump files were renamed. See “Virtual Machine Dumps in an XC Environment” on page 11 for more information.
5. You also have the capability (END) to back up to the previous screen (list of matches or Command input menu) in order to look at another record.

Subcommands:

QUIT
This subcommand returns control to CMS or the previous command environment.

END
This subcommand returns control to the previous level either a list of matches or the command menu. If you return to the command menu, the previous selection criteria is blanked out, and you can enter a new search criteria.

Search
This subcommand extracts the primary symbol strings from the current record and uses them as search criteria for duplicate records. Only the primary symbol strings are used; the date and time range and CPU serial number are defaulted to blank. The new list contains the selected symptom record identified by an asterisk to the left of the screen.

Respecify
This subcommand returns to the command menu and lets you either broaden or narrow the previous search criteria. The previous search operand will be filled in as the default, and you can respecify the search operands.

Forward
This subcommand scrolls the file toward the end of the file.

Backward
This subcommand scrolls the file toward the beginning of the file.

Chapter 4. Macro Subcommands

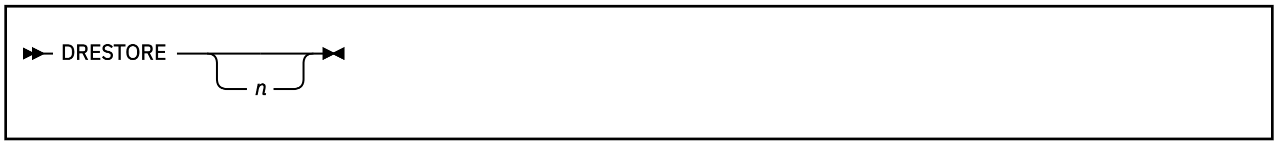
PI

This chapter describes the DUMPSCAN subcommands that can only be used within a macro. The function of these subcommands is summarized in Table 7 on page 39. The syntax and usage information for these subcommands is presented following the table.

Table 7. Subcommands Used Only within a MACRO

Subcommand	**Description**
DRESTORE	Use the DRESTORE subcommand to restore the settings of the DUMPSCAN variables to the values they had when the DSAVE subcommand was last entered.
DSAVE	Use the DSAVE subcommand to save the settings of various DUMPSCAN variables until a subsequent DRESTORE subcommand is entered.
DVFSTACK	Use the DVFSTACK subcommand to direct DUMPSCAN subcommand output to the program stack.
FINDStrg	Search for a particular string of data in the dump.
INIT	Puts the name of the dump and the dumptype into the session file.
NOTE	Send text output to the Dump Viewing Facility to be displayed on the terminal, printer, or both
READStrg	Read data from the dump, through a direct or indirect address.
SCAN	Process a PF key assignment or command string to the system product interpreter.

DRESTORE Subcommand



Purpose

Use the DRESTORE subcommand to restore the settings of the DUMPSCAN variables to the values that existed when the DSAVE subcommand was last entered.

Operands

n
is an integer greater than 0 indicating the number of the saved variable buffer to restore. If not specified, the DUMPSCAN settings from the last DSAVE are used.

Usage Notes

1. If the number of the saved variable buffer requested is not in the stack of saved settings, the DUMPSCAN settings remain unchanged.
2. All numbers of settings after the requested number are deleted.
3. Refer to the DSAVE subcommand for a list of settings affected by the DRESTORE subcommand.

Messages and Return Codes

Return Code	Explanation
0	Successful completion
4	No DSAVE has been issued
8	The requested buffer was not found
16	Invalid operand, no extra operands are allowed
32	Internal error

DSAVE Subcommand

```
▶▶─ DSAVE ─▶◀
```

Purpose

Use the DSAVE subcommand to save the settings of various DUMPSCAN variables until a subsequent DRESTORE subcommand is entered.

Usage Notes

The following are saved:

1. DVFSTACK setting
2. Address of the last addresses displayed
3. SELECT subcommand settings
4. TRACE subcommand, first and last addresses of the last display
5. The last character strings used as an operand in the LOCATE subcommand
6. Whether PRINT was ON or OFF

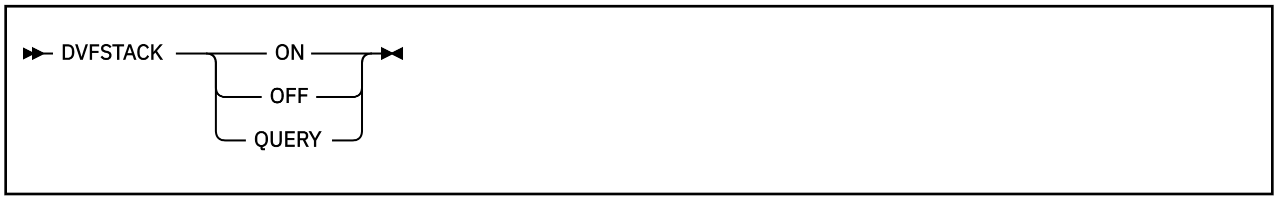
Messages and Return Codes

Return Code
  Explanation

0
  Unable to complete request, out of storage, or an invalid operand was specified.

n
  Number of the saved variables buffer just created.

DVFSTACK Subcommand



Purpose

Use the DVFSTACK subcommand to direct DUMPSCAN subcommand output to the program stack.

Operands

ON
: is a keyword operand that directs DUMPSCAN subcommand output to be put in the program stack.

OFF
: is a keyword operand that resets the DVFSTACK ON invocation, so DUMPSCAN subcommand output is directed to the session VIEW*nnnn* file. This setting is always restored when control returns to the DUMPSCAN command line.

QUERY
: is the keyword operand that does not change the current DVFSTACK setting but returns the setting of DVFSTACK as a return code.

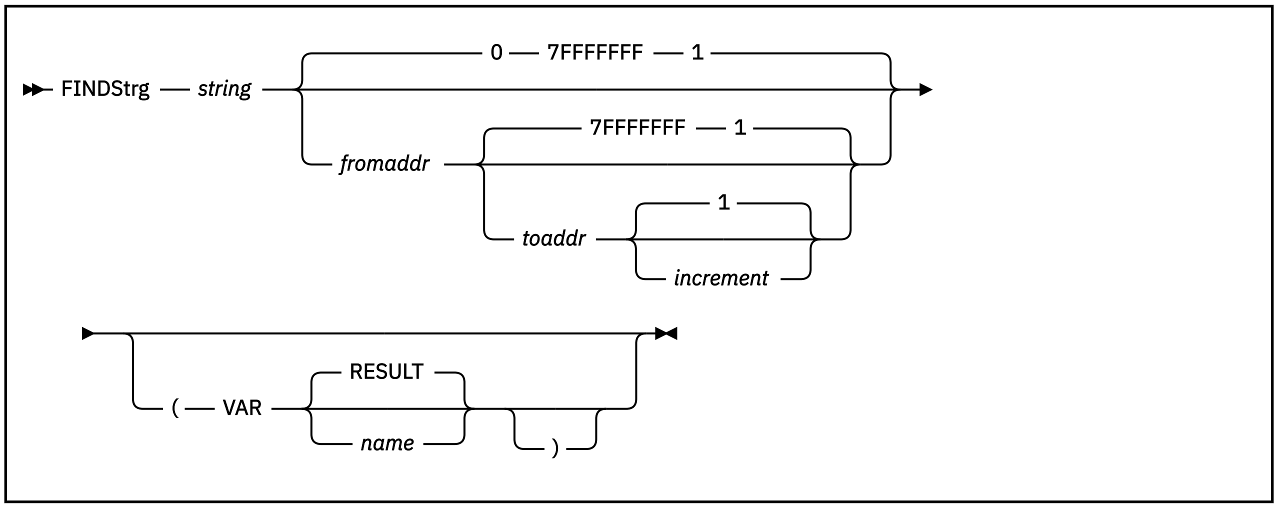
Usage Notes

1. This macro (subcommand) can only be executed from a macro. An error message will be displayed if entered from the command line.
2. When DVFSTACK ON is in effect, all messages issued by DUMPSCAN subcommands are directed to the program stack.
3. When DVFSTACK ON is in effect, and a DUMPSCAN subcommand is issued, the redisplay line (--->) is neither returned to the program stack nor displayed in the VIEW*nnnn* file.
4. When DVFSTACK ON is in effect, and the macro issues a subsequent DVFSTACK ON, the setting is unchanged and a return code is set for the calling macro. The same scenario applies for DVFSTACK OFF.
5. It is the responsibility of the macro to manage its own program stack resources; otherwise, unpredictable or erroneous results may occur during the DUMPSCAN session.

Messages and Return Codes

Return Code	Explanation
0	Successful execution, the state of DVFSTACK was changed to ON or OFF
2	You issued DVFSTACK QUERY or DVFSTACK ON; and DVFSTACK ON was previously set; The setting is unchanged
4	You issued DVFSTACK QUERY or DVFSTACK OFF; and DVFSTACK OFF was previously set; The setting is unchanged
8	Invalid operand or required operand missing

FINDSTRG Subcommand



Purpose

Use the FINDSTRG subcommand to search for a particular string of data in the dump while you are still in an EXEC. If found, the address of the string is returned in a REXX variable.

Operands

string
is a 1- to 128-character (1- to 64-byte) hexadecimal string for which you are searching.

fromaddr
is the 31-bit (1- to 4-byte) hexadecimal starting address for the search. If not specified, this defaults to start at location 0. Leading zeros are not required.

toaddr
is the 31-bit (1- to 4-byte) hexadecimal ending address for the search. If not specified, this defaults to end at location 7FFFFFFF. Leading zeros are not required.

increment
is a 1- to 4-digit hexadecimal number to change the current address after each match attempt.

Options

VAR
is a keyword operand indicating that the following is the user-specified REXX variable name.

RESULT
is the default name of the REXX variable if you do not specify a name.

name
is a 1- to 8-character user-specified name of a REXX variable where the results of the FINDSTRG subcommand will be placed.

Usage Notes

1. This subcommand can only be executed from a macro. An error message is issued if entered from the command line.
2. Unlike the LOCATE subcommand which accepts either EBCDIC characters or hexadecimal digits, the FINDSTRG subcommand accepts only hexadecimal digits. If EBCDIC data such as a user ID needs to be located, it must be converted to hex first.

3. The start of the string must be within the address range specified by the *fromaddr* and *toaddr* addresses. If the *fromaddr* and *toaddr* addresses are not specified, they will default to the beginning and ending of the dump.
4. In order to specify an *increment*, both the *fromaddr* and *toaddr* addresses must be specified.
5. The address of the first byte of the string, if found, is placed in a REXX variable (RESULT or the user-specified name).
6. If you want to look for multiple occurrences of a string within a dump, you must update the *fromaddr* after each match. The reuse or = subcommands do not apply to this subcommand.
7. The valid increment range is from X'1' to X'1000'.

Messages and Return Codes

Return Code	Explanation
0	Successful execution
8	String not found
16	Invalid operands
20	Internal error

INIT Subcommand

```
►► INIT ►◄
```

Purpose

Use the INIT subcommand to put the name of the dump and the dump type into the session file.

Usage Notes

1. INIT is only valid from within a macro.
2. A macro can use this subcommand to obtain the dump type.
3. Messages 200 and 401 are put into the session file.

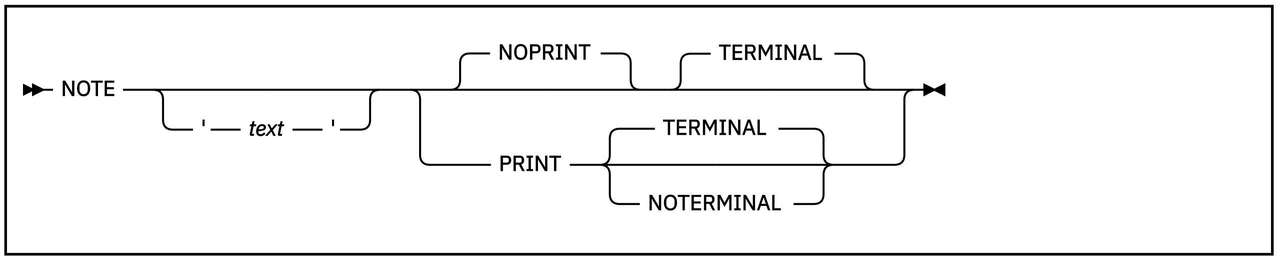
Messages and Return Codes

HCSDSS200I
PROCESSING FILE *filename filetype fm*1

HCSDSS401I
READY, DUMP TYPE IS *dumptype*

Return Code	**Explanation**
0	Successful execution

NOTE Subcommand



Purpose

Use the NOTE subcommand to send text output to the Dump Viewing Facility to be displayed on the terminal, the printer, or both.

Operands

text
: is the output to be displayed. This includes any leading blanks. The maximum length of the text is 80 bytes. If no text is specified, a blank line is printed or displayed according to the options selected or defaulted to. Beginning and ending quotation marks are mandatory if text is specified.

PRINT
: indicates that the text should be sent to the virtual printer. This operand may not be specified in conjunction with the NOPRINT operand.

NOPRINT
: indicates that the text should not be sent to the virtual printer. This operand may not be specified in conjunction with the PRINT or NOTERMINAL operands. This operand is the default.

TERMINAL
: indicates that the text should be displayed on the terminal. This operand may not be specified with the NOTERMINAL operand. This operand is the default.

NOTERMINAL
: indicates that the text should not be displayed on the terminal. This operand may not be specified in conjunction with the TERMINAL or NOPRINT operands.

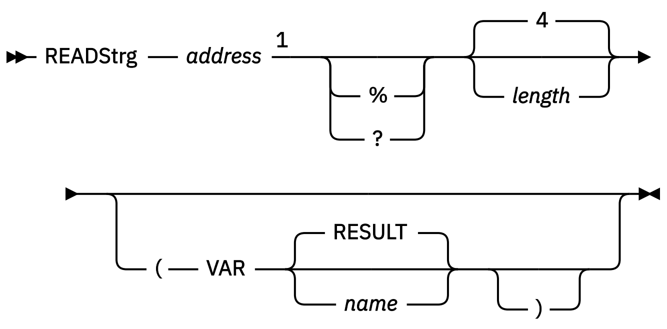
Usage Notes

1. This subcommand is only valid when issued from a macro.
2. Text longer than 80 bytes will result in a return code of 8.

Messages and Return Codes

Return Code	Explanation
0	Successful execution
8	Invalid conditions such as conflicting operands, a missing quotation mark, or text longer than 80 bytes
500	Virtual printer error (A message indicating this error is displayed on the terminal.)

READSTRG Subcommand



Notes:

[1] Do not put blanks between the operands and special characters.

Purpose

Use the READSTRG subcommand to read data from the dump while you are in a macro. You can specify the actual or an indirect address. The data at that address is returned in a REXX variable.

Operands

address
: is the 31-bit (1- to 4-byte) hexadecimal address from which the data is to be retrieved in the dump. Leading zeros are not required.

%
: specifies a 24-bit indirect address. A word (4 bytes) of storage at the specified address is read from the dump and used as the basis for a second read. The data at the second address is returned to the macro.

?
: specifies a 31-bit indirect address. A word (4 bytes) of storage at the specified address is read from the dump and used as the basis for a second read. The data at the second address is returned to the macro.

length
: is an optional operand. It is a 1- to 4-digit nonzero hexadecimal number indicating the length in bytes to be returned to the macro. The valid range is from X'1' to X'1000'. Four is the default length of the data to be returned.

Options

VAR
: is a keyword operand indicating that the following is the user-specified REXX variable name.

RESULT
: is the default name of the REXX variable that is used if the user does not specify a name.

name
: is a 1- to 8-character user-specified name of a REXX variable where the results of the READSTRG subcommand are placed.

Usage Notes

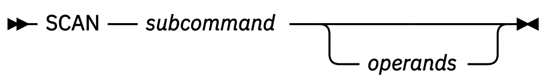
1. This subcommand can only be executed from a macro. An error message is issued if it is entered from the command line.

2. The dump data is translated to EBCDIC and then returned to the macro in a REXX variable (that is, in the parameter RESULT or the user-specified name).
3. If only partial data is available in the dump, READSTRG returns only the available data, in which case the user should check the length of REXX variable being used.

Messages and Return Codes

Return Code	Explanation
0	Successful execution
4	Partial data returned
8	Page not dumped
16	Invalid operand
20	Internal error

SCAN Subcommand



Purpose

Use the SCAN subcommand to process a PF-key assignment or a command string from the system product interpreter.

Operands

subcommand
is any valid DUMPSCAN subcommand.

operands
includes any operands for the DUMPSCAN subcommand you requested.

Usage Notes

1. Use the SCAN service to assign Dump Viewing Facility functions to PF keys. For more information, see "Assigning Program Function Keys to DUMPSCAN Subcommands" on page 17.
2. Use the system product interpreter command ADDRESS to affect a temporary or permanent change to the destination of commands. The SCAN environment is addressable from any system product interpreter macro during the DUMPSCAN session. For more information, see "What Is an Environment?" on page 13. For more information, see "What Is an Environment?" in *z/VM: Dump Viewing Facility*.
3. DUMPSCAN initializes SCAN by loading DVSCAN SCAN as SCAN EXEC into storage using EXECLOAD. For further information on EXECLOAD see *z/VM: CMS Commands and Utilities Reference*.

PI end

Chapter 5. DUMPSCAN Subcommand Reference

This chapter contains reference information for the DUMPSCAN subcommands used to interactively view data from a dump. The DUMPSCAN macro services are described in Chapter 4, “Macro Subcommands,” on page 39.

Table 9 on page 51 lists all the subcommands for DUMPSCAN and the functional category of each. The minimum truncation for each command is indicated by the uppercase letters in the subcommand column. The definition of the functional categories is:

COMMON
All dumps

VM
Virtual machine dump

CMS
CMS dump

GCS
GCS dump

TSAF
TSAF dump

AVS
AVS dump

SFS
SFS dump

RSCS
RSCS dump

Note: SFS includes the coordinated resource recovery facility.

The following table identifies the supported dump types that can have component unique DUMPSCAN subcommands which are not described in this document issued against them and where you can find more information on how to use them:

Table 8. Supported Dump Types

VM Dump Type	Reference
PVM	*VM/SP Pass-Through Facility Logic*
RSCS	*z/VM: RSCS Networking Diagnosis*
CICS/VM	*CICS/VM Problem Determination Guide*

Table 9. Subcommands for DUMPSCAN

Subcommand	Functional Category	Description
(null line)	COMMON	Continues the previous CHAIN, LOCATE, LOCATEUP, BACKWARD, or FORWARD subcommand.
+ (plus symbol) - (minus symbol)	COMMON	Moves you forward through hexadecimal; moves you backward through hexadecimal.
& (ampersand) or &name	COMMON	Assigns symbolic names to subcommands.
? (question mark)	COMMON	Displays the last subcommand entered.

Table 9. Subcommands for DUMPSCAN (continued)

Subcommand	Functional Category	Description
= (equal symbol)	COMMON	Re-executes the previous DUMPSCAN subcommand.
ACClist	VM	Displays information about the VM data spaces that contributed data to the dump.
Aregs	COMMON	Displays access registers for a specified processor.
ASid	VM	Identifies which address space contributed information to a dump file.
Backward	COMMON	Scrolls backward through hexadecimal data or trace entries.
BLock	COMMON	Formats control blocks within a dump.
CHain	COMMON	Displays the chain of control block addresses.
CMS	COMMON	Allows an application to enter the CMS subset mode.
CMSPoint	CMS	Displays the formatted contents of pointers from CMS NUCON.
CMSVIEW	CMS	Displays CMS control blocks, chains, and trace data.
CMSVIEW TRACE	CMS	Displays the CMS trace data.
CPU	COMMON	Displays the address and prefix register values for each processor.
Cregs	COMMON	Displays the control registers for a specific processor.
Display	COMMON	Displays dump data in both hexadecimal and EBCDIC.
DOSPoint	CMS	Displays the formatted contents of five pointers used by DOS simulation.
DUMPID	VM	Displays the dump identifier.
DUMPScan	COMMON	Changes DUMPSCAN display to a different dump file.
END	COMMON	Ends DUMPSCAN execution and returns the application to CMS.
FDISPlay	TSAF	Displays data control blocks, tables, and arrays important to the TSAF virtual machine.
FINDMod	COMMON	Displays the displacement and module name or entry point, given an address; or displays an address, given a module name or entry point.
Forward	COMMON	Scrolls forward through hexadecimal data or trace entries.
FORMAT	VM	Displays or changes the dump type (that is, format) of a dump file.
GDISPLAY	AVS	Displays control blocks important to the AVS virtual machine, and the module name and module address for APPC/VM VTAM.
Gregs	COMMON	Displays general-purpose registers for the specified processor.
HC	COMMON	Resolves hexadecimal calculations and algebraic expressions.
HELP	COMMON	Displays individual DUMPSCAN help files or the menu that lists all the DUMPSCAN help files.
HX	COMMON	Ends that particular dump viewing session.
IUcv	GCS, AVS, RSCS	Displays all entries in the IUCV path table.

Table 9. Subcommands for DUMPSCAN (continued)

Subcommand	Functional Category	Description
Locate	COMMON	Locates the next occurrence of a hexadecimal or character string in a dump.
LocateUp	COMMON	Locates the previous occurrence of a hexadecimal or character string in a dump.
OSPoint	CMS	Displays the formatted contents of three pointers used in OS simulation.
Print or PRT	COMMON	Directs output of the subcommand to the printer.
QUIT	COMMON	Ends that particular dump viewing session.
Regs	COMMON	Displays registers, clocks, timer, and program status words for a specific processor.
Scroll U or ScrollU	COMMON	Repeats the most recent TRACE subcommand with an adjusted address.
SYMPtom	COMMON	Displays symptom record data.
TACtive	GCS, AVS, RSCS	Displays a task's active program list.
TIMediff	COMMON	Displays the difference in time between two TOD clock values.
TLoadl	GCS, AVS, RSCS	Displays the task load list.
TODCLK	COMMON	Displays the date and time for a specified hexadecimal TOD clock value.
Trace	AVS, SFS, TSAF	Displays trace table entries.
TSab	GCS, AVS, RSCS	Displays the subpool map and storage owned by a task.
VMLoadl	GCS, AVS, RSCS	Displays information about all programs loaded in a virtual machine.
Xedit	COMMON	Passes the command line to XEDIT for execution.

Each of the following subcommand descriptions include the subcommand syntax, keywords, operands, and options. Where applicable, the descriptions also include usage notes, responses, messages, and examples of the output to be expected from each subcommand.

The same notational conventions apply to the DUMPSCAN subcommands as described in Chapter 3, “Command Reference,” on page 21.

General Usage Notes:

1. All addresses in this chapter refer to real addresses unless otherwise specified.
2. All dump storage addresses are 31-bit, 4-byte addresses containing up to eight hexadecimal characters unless otherwise specified. Leading zeros can be omitted. For example, if you want to enter address 00012F31, you may enter 00012F31 or 12F31.
3. All processor addresses (*cpuaddr*) are 1- to 4-byte hexadecimal digits.
4. All device numbers are up to four hexadecimal digits.
5. Logical real device numbers are up to four hexadecimal digits and are prefixed with the letter L.
6. Unless otherwise specifically noted, wherever the term CP abend dump is mentioned, it can be assumed that the command output will be the same for a snapdump because the two dumps are identical in content.

Null Line Subcommand

Purpose

Use the null line subcommand to reissue the previous CHAIN, LOCATE, LOCATEUP, BACKWARD, or FORWARD subcommands.

Usage Notes

1. Pressing Enter with no data entered repeats the previous CHAIN, LOCATE, LOCATEUP, FORWARD, or BACKWARD subcommands with an updated address.
2. Entering a null line is valid for the CHAIN subcommand only if the number of control blocks exceeds 4096. A message is issued when there are more than 4096 control blocks. Entering a null line continues the chain presentation starting with the last address displayed.
3. The running total of all members found is displayed in the output of the reissued CHAIN subcommand.

Responses

Using the null line command re-executes the CHAIN, LOCATE, LOCATEUP, BACKWARD, or FORWARD commands; a full screen of the appropriate data is displayed.

+ and - Subcommands

```
►►─ + ── increment ─►◄

►►─ - ── decrement ─►◄
```

Purpose

Use the + and - subcommands to adjust the address pointer and reissue the DISPLAY subcommand.

Operands

increment
is the hexadecimal number to be added to the address pointer of the last displayed subcommand entered.

decrement
is the hexadecimal number to be subtracted from the address pointer of the last displayed subcommand entered.

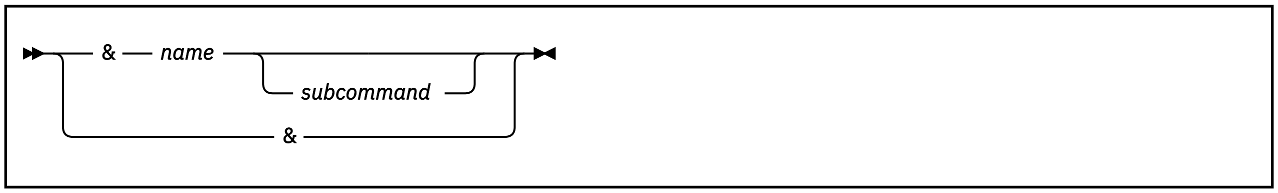
Usage Notes

1. Use the + and - subcommands after entering the displaying and scrolling subcommands of the Dump Viewing Facility: BACKWARD, DISPLAY, FINDMOD, FORWARD, LOCATE, and LOCATEUP.
2. The increment value has no upper limit. If the resulting address is outside the dump's range, the system displays an error message.
3. These subcommands do not wrap the screen.
4. If you specified the OFFSET operand on the previous DISPLAY subcommand, entering the + or - subcommand results in continued display of offsets.

Responses

If enough data remains in the dump, the system displays an entire screen of dump data. The current line position moves to the calculated address.

&name Subcommand



Purpose

Use the &*name* subcommand to create a table of frequently used subcommands that may be invoked by another name, or to invoke a subcommand by its other name.

Operands

name
is the symbolic name you give to the subcommand expression entered in the &name table. The name portion of this subcommand may be up to 7 characters in length and must be preceded by the ampersand.

subcommand
is the entire syntax of the subcommand including any operands specified. Entering &*name* without any operands invokes the subcommand.

&
Entering an ampersand (&) alone lists all the table entries.

Usage Notes

1. When entering data into the &name table, you may not enter another &*name* subcommand. For example,

   ```
   &name1 &name2
   ```

   is not allowed.
2. If you try to invoke an &*name* that is not in the table, DUMPSCAN displays an error message.
3. The subcommand in the table is not checked for validity until it is invoked by entering &*name*. Only then are errors detected by the appropriate subcommand processor.
4. The PRINT subcommand is not allowed in the &name table.
5. All entries into the &name table are limited to 8 characters for each operand. The &*name* subcommand plus eight operands may be entered, that is, &VM VMDBK *operand2 operand3...... operand8*.
6. Up to 64 operands, including the symbolic names (&*name*), may be contained in the &name table at any one time. The number of symbolic names you are limited to is determined by the number of operands used per subcommand. You can assign more subcommands with three operands (21) than you can with seven operands (nine).
7. If the LOCATE subcommand is placed in the &name table, the maximum string of 8 characters includes the hexadecimal identifier X, the hexadecimal characters, and the quotation marks (for example, X'13AB4').

Examples

The &*name* subcommand is useful for command strings that are used constantly. It lets you shorten a command string to one symbolic name.

Figure 8 on page 57 illustrates a sequence of six &name entries being made in the &name table from the command line.

```
====> &dn -1000
====> &up +1000
====> &lo locate feibm 0 7fffffff
====> &d display 6a000 100 offset
====> &ch chain 1000 600 f4000
====> &d display 6000 100
```

Figure 8. A Sequence of &name Commands

After all six entries are made, you can check the &name table by entering the &name table list subcommand &. This displays the &name table as shown in Figure 9 on page 57. The subcommand entered is:

```
&
```

```
&DN -1000
&UP +1000
&LO LOCATE FEIBM 0 7FFFFFFF
&D DISPLAY 6A000 100 OFFSET
&CH CHAIN 1000 600 F4000
&D DISPLAY 6000 100
```

Figure 9. Listing the &name Table

Any time you wish to execute a command string in the &name table, enter the symbolic name corresponding to the desired command. The command in the &name table is processed as if it were just entered manually.

Figure 10 on page 57 shows the symbolic command *&d* being entered, and the resulting display of the corresponding subcommand, DISPLAY 6000 100.

The subcommand entered to produce the output shown in Figure 10 on page 57 is:

```
&d
```

```
DISPLAY 6000       100             OFFSET
      0000  00000000  00000000  00000000  00000000 F6 *................*
      0010  00000000  00000000  00000000  00000000    *................*
      0020  00000000  00000000  00000000  00000000    *................*
      0030  00000000  00000000  00000000  00000000    *................*
      0040  00004040  40400000  00000000  00000000    *..    ..........*
      0050  00000000  00000000  00000000  00000000    *................*
      0060  00000000  00000000  C4D4E2D4  D4D4D5D5    *........DMSMMMNN*
      0070  D5D34040  40404040  40404040  40404040    *NL              *
      0080  40404040  40404040  40404040  40404040    *                *
      0090  40404040  40404040  40404040  40404040    *                *
      00A0  40404040  40404040  40404040  40404040    *                *
```

Figure 10. Executing the DISPLAY Subcommand Using a Symbolic Name

Responses

1. If &*name* is entered, the response is from the subcommand executed.
2. If & is entered, a list of the current entries in the &name table is displayed.
3. If &*name subcommand* is entered, the ready response indicates the subcommand has been added to the &name table.

? Subcommand

```
►►─ ? ─►◄
```

Purpose

Use the ? subcommand to display the last subcommand entered.

Usage Notes

1. The subcommand displayed as a result of a question mark (?) can be re-executed by pressing Enter. You can also modify the command before entering it again.
2. Successive execution of the ? subcommand will display the previous subcommands.
3. A synonym cannot be defined for the ? subcommand.
4. The ? subcommand can be assigned to a PF or a PA key.
5. Anything following a ? is ignored except another ?. Multiple question marks can be specified to retrieve previous subcommands.
6. The results of the execution of the equal (=) subcommand may not be identical with the results of combining the ? subcommand and the Enter key. The = subcommand executes the last valid Dump Viewing Facility subcommand.

Responses

The system displays the last command line entered from the terminal.

= Subcommand

```
▶▶— = —▶◀
```

Purpose

Use the = subcommand to re-execute the last successful DUMPSCAN subcommand.

ACCLIST Subcommand

```
►► ACClist ►◄
```

Purpose

Use the ACCLIST subcommand to display information about the VM data spaces that contributed data to the dump.

Usage Notes

1. This subcommand is valid for virtual machine dumps.
2. ALET X'00000000', which indicates the primary address space, is not displayed by the ACCLIST subcommand.
3. ACCLIST searches the accessed disks for a file with a file name matching the current dump file's and a file type of DUMP0001. Renaming files or using duplicate names can cause ACCLIST to fail or to return information about the wrong dump. See "Virtual Machine Dumps in an XC Environment" on page 11 for notes on renaming dump files from XC virtual machines.

Responses

The response to the ACCLIST subcommand looks like this:

```
OWNING ADDRESS SPACE IDENTIFIER = USER1:BASE

ALET       ACC    ASIT                SPACE IDENTIFICATION
01000002   R/W    007D730000000004    USER1:USER1SP1
01000003   R/W    007D734000000004    USER1:USER1SP2
01000004   R/O    **REVOKED**
01000005   R/O    007D738000000004    USER1:USER1SP3
01000006   R/O    007D73C000000004    USER1:USER1SP4
01000007   R/W    007D72C000000004    USER2:USER2SP1

6 TOTAL ENTRIES:  5 VALID, 1 REVOKED, 1017 UNUSED
```

ACCLIST returns information in these fields:

OWNING ADDRESS SPACE IDENTIFIER
is the identifier of the address space in which the access lists reside.

ALET
is the access list entry token corresponding to the access list entry. It is eight hexadecimal digits long.

ACC
indicates the access permitted by the ALET, either read/write or read-only.

ASIT
is the address space identification token. It is specified as 16 hexadecimal digits.

When the value in the ASIT field is **REVOKED**, the ALET previously designated an address space, but access to that address space has been revoked.

SPACE IDENTIFICATION
is the space identifier for the data space, in the form *owner*:*space_name*, where *owner:* is the user ID that owns the address space and *space_name* is the name of the address space. The space name is a string of alphanumeric and national characters, 1 to 24 characters long.

The SPACE IDENTIFICATION field is left empty for revoked ALETs.

TOTAL ENTRIES
is the total number of valid or revoked entries in the access list.

VALID
: is the number of ALETs that designate address spaces and that can be used to access data in those address spaces.

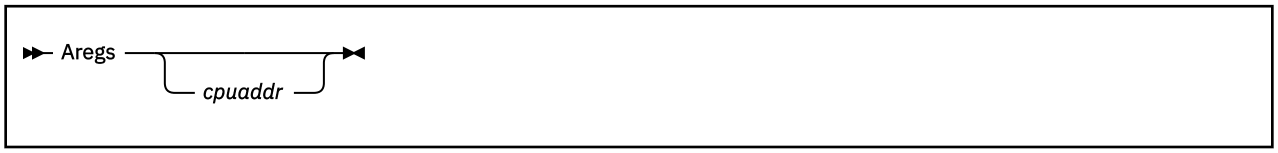
REVOKED
: is the number of ALETs that previously designated available address spaces but that can no longer be used to access data.

UNUSED
: is the number of entries in the access list that remain available for use.

Messages and Return Codes

Return Code	**Explanation**
0	Successful completion
8	Access list entries are in DUMP0001, which is not accessed
12	Access list is empty
16	Access list is not available for SPACE *owner*:*space_name*
104	Internal processing error

AREGS Subcommand



Purpose

Use the AREGS subcommand to display access registers for a specified processor.

Operands

cpuaddr
is a 1-to-4-digit hexadecimal number specifying the physical CPU address for which the general registers are to be displayed.

Usage Notes

1. If the *cpuaddr* operand is not specified, it defaults to the CPU on which the CP VMDUMP command was entered for the virtual machine.
2. Use the CPU subcommand to obtain the CPU addresses in the dump.

Examples

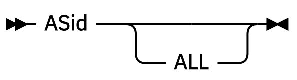
Figure 11 on page 62 shows the output of an AREGS subcommand. The subcommand entered is:

```
Aregs
```

```
CPU ADDRESS - 0000
ACCESS REGS  0 - 15
     00000000 00010000 00000036 803E8088  003E7F30 00FCD298 00000037 00FD0748
     0031CD90 00379900 00FCE1F8 00F8C000  00378900 00FC9E80 8037901C 00000000
```

Figure 11. Sample Output of an AREGS Subcommand

ASID Subcommand



Purpose

Use the ASID subcommand to display address space information.

Operands

ALL
causes address space information from all related dump files to be displayed. The default is to display information about the current dump file.

Usage Notes

1. This subcommand is valid for virtual machine dumps.
2. If ALL is specified, an attempt is made to read the information in other dump files with the same file name and a file type of DUMPnnnn.
3. ASID searches the accessed disks for a file with a file name matching the current dump files and a file type of DUMPnnnn. Renaming files or using duplicate names can cause ASID to fail or to return information about the wrong dump. See "Virtual Machine Dumps in an XC Environment" on page 11 for notes on renaming dump files from XC virtual machines.

Responses

If six data spaces were dumped, the response to

```
ASID ALL
```

might look like this:

```
FILETYPE        ASIT                    SPACEID                 FORMAT
DUMP0001  007D724000000001  USER1:BASE                          CMS
DUMP0002  007D730000000004  USER1:USER1SP1                      CMS
DUMP0003  007D734000000004  USER1:USER1SP2                      CMS
DUMP0004  007D738000000004  USER1:USER1SP3                      CMS
          (DUMP NOT AVAILABLE)
DUMP0006  007D73C000000004  USER1:USER1SP5                      CMS
```

ASID returns information in these fields:

FILETYPE
The file type of the dump file containing this information. It is eight characters long.

ASIT
is the address space identification token. It is 16 hexadecimal digits long.

`(DUMP NOT AVAILABLE)` is displayed in the ASIT field for each of the related dumps that is not accessible to the Dump Viewing Facility.

`(ADDRESS SPACE INFORMATION NOT AVAILABLE)` is displayed in the ASIT field for any of the related dumps when the address space information is not accessible to the Dump Viewing Facility.

SPACEID
is the address space identifier in the form *owner*:*space_name*, where *owner:* is the user ID that owns the address space and *space_name* is the name of the address space. The space name is a string of

alphanumeric and national characters, 1 to 24 characters long. The SPACEID field is empty when the dump or the address space information is not available.

FORMAT
The type of virtual machine dump. It is 1 to 8 characters long. The FORMAT field is empty when the dump or the address space information is not available.

Messages and Return Codes

Return Code	Explanation
0	Successful completion
4	An unrecognized operand was specified
104	Internal processing error

BACKWARD Subcommand

```
►►─ Backward ─►◄
```

Purpose

The BACKWARD subcommand scrolls backward toward the lowest address in the dump.

Usage Notes

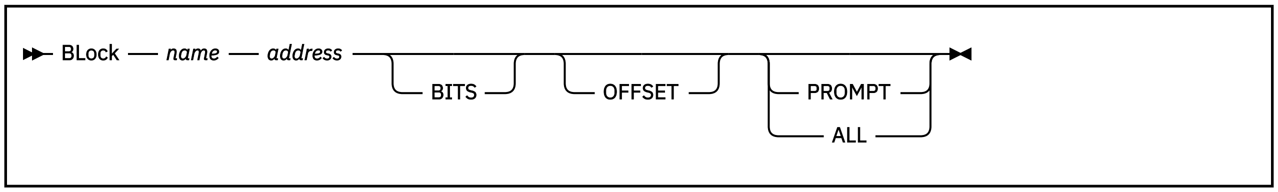
1. The BACKWARD subcommand can be used after you enter the DISPLAY, LOCATE, FINDMOD, TRACE, or other scrolling subcommands.
2. The BACKWARD subcommand may be reentered by pressing Enter (*null line* subcommand).
3. For scrolling forward to the highest address in the dump, see the FORWARD subcommand later in this chapter.
4. If you entered the OFFSET operand on the previous DISPLAY subcommand, then the BACKWARD subcommand continues to display data using the specified offsets. Your terminal screen continues to display the original storage address requested.
5. The BACKWARD subcommand does not wrap the screen.
6. You cannot display data with offsets below 0.

Responses

One full screen of data is presented in both hexadecimal and EBCDIC. For example, if your screen displays 19 lines, the data at the top of the screen (first line) is hexadecimal 130 bytes from the last address displayed.

When scrolling after the TRACE subcommand, the format of the next screen is identical with the screen when TRACE was entered. For example, if the previous TRACE subcommand was for FORMAT output, scrolling continues with formatted output.

BLOCK Subcommand



Purpose

Use the BLOCK subcommand to format control blocks within a dump. You can format the entire block or selected fields. You can also request that a predefined subset of high-interest fields be formatted.

Use the BLOCKDEF utility command to generate control block format files used by the DUMPSCAN BLOCK subcommand, and to generate control block information print files for users. For more detailed information regarding this utility, see the "BLOCKDEF Utility Command" on page 171 in Appendix C, "Dump Viewing Facility Utilities," on page 163.

For an example which illustrates the necessary steps needed to set up a control block file for the BLOCK subcommand see "Adding Block Definition Files" on page 146 in Appendix A, "Using Attachment Interfaces," on page 139.

Operands

name
is the 1- to 8-character name of the control block to be formatted.

address
is a 1- to 8-digit hexadecimal address indicating the storage location of the control block.

BITS
is a keyword indicating that the bits within a byte should be formatted when possible.

Note: If not specified, then only information down to the byte level is formatted.

OFFSET
is a keyword indicating that the display should be formatted using relative offsets from the start of the control block, instead of actual addresses.

PROMPT
is a keyword indicating that the user would like to be prompted for the field names to be displayed.

ALL
is a keyword indicating that all fields within the control block are to be formatted.

Usage Notes

1. The Dump Viewing Facility BLOCK subcommand is capable of mapping any control block for any type of dump. See section "Block Table Architecture" on page 145 in Appendix A, "Using Attachment Interfaces," on page 139 for information concerning how to do this.
2. If you do not specify either ALL or PROMPT, only fields that are marked as "default" in the table for the control block are displayed.
3. BLOCK does not verify that the control block name provided is valid for the address given. If the user gives BLOCK the wrong address, BLOCK simply maps the storage into the control block definition as if the address were correct.
4. When using the PROMPT function of BLOCK, you can display a selected group of fields and then discover that additional fields need to be displayed. In this instance you need not retype all of the fields entered previously. When prompted for the fields to be displayed, you have the option of reusing the old fields and having the new fields added to the display. This is accomplished by entering an equal

sign (=) followed by the name of the new fields. BLOCK redisplays the previous fields, followed by the new ones. The following is an example of the use of PROMPT with an equal sign.

On a prompt for field names to be displayed, you have entered the following:

```
field1 field2 field3 field4
```

You now want to add fields 5 and 6 to the display. After reentering the BLOCK command with the PROMPT keyword, you are again prompted for the field names to be displayed, and enter the following:

```
= field5 field6
```

BLOCK displays the fields in the order that you entered them. The program does not try to order the fields, nor does it check for duplicate fields. BLOCK only verifies that a name entered by the user actually exists in the control block definition.

5. You can flag any field as a default field with the exception of BIT subrecord fields. If a BIT field is flagged as a default field and the BITS keyword was specified, the bits are displayed. If just the BIT subrecord field is flagged, the flag is ignored.

Examples

The following examples demonstrate how the various BLOCK operand and keyword combinations provide you with the control block data you need.

If you enter **block userblok 20000 all**, the response is as shown in Figure 12 on page 67.

```
              BLOCK  USERBLOK  AT  LOCATION 00020000

     ADDR/OFF  NAME      CONTENTS          DESCRIPTION

     00020000  USERFLGA  A2                EVENT STATUS FLAGS
     00020001  USERFLGB  3E                EVENT STATUS FLAGS #2
     00020002  *         0000              RESERVED
     00020004  USERLINK  00389008          LINK POINTER
     00020008  USERCBID  'USBK'            CONTROL BLOCK IDENTIFIER
     0002000C  USERREGF  00023810          SAVED RETURN CODE FROM CALL
     00020010  USERREGE  00650101          SAVED REGE FROM PRIOR CALL
     00020014  USERTIME  6DFC83E94AA3CB13  TIME OF DISPATCH TO CPU 1
                                           DISPATCHER/SCHEDULER ROUTINE
     0002001C  USERFLGC  A1                LOCK FLAGS
```

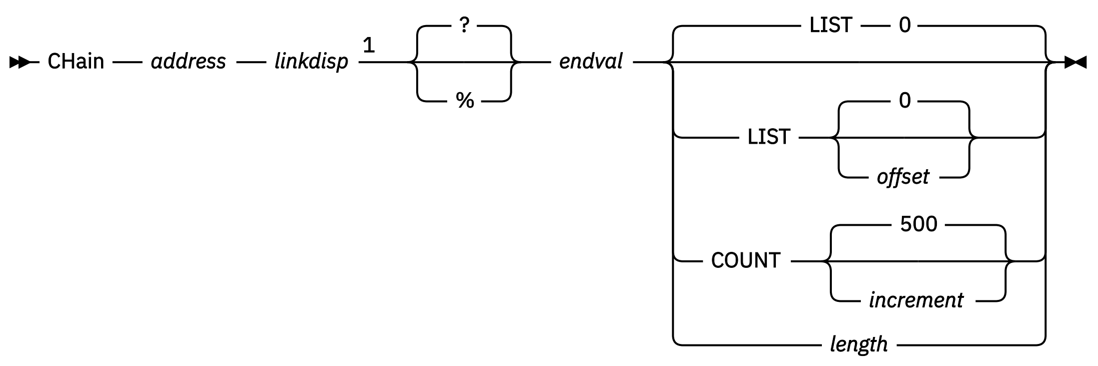
Figure 12. Sample Output of the BLOCK Subcommand with the ALL Keyword

If you enter **block userblok 20000 bits offset**, the response is as shown in Figure 13 on page 68.

```
          BLOCK  USERBLOK  AT  LOCATION 00020000

     ADDR/OFF  NAME      CONTENTS          DESCRIPTION

     00000000  USERFLGA  A2                EVENT STATUS FLAGS
               USERBIT1 1...  ....         I/O IN PROGRESS
               USERBIT2 .0..  ....         DEACTIVATE STARTED
               USERBIT3 ..1.  ....         SESSION ENDED
               USERBIT4 ...0  ....         PURGE Q REQUESTED
     00000001  USERFLGB  3E                EVENT STATUS FLAGS #2
     00000002  *         0000              RESERVED
     00000004  USERLINK  00389008          LINK POINTER
     00000008  USERCBID  'USBK'            CONTROL BLOCK IDENTIFIER
     0000000C  USERREGF  00023810          SAVED RETURN CODE
                                           FROM CALL
     00000010  USERREGE  00650101          SAVED REGE FROM
                                           PRIOR CALL
     00000014  USERTIME  6DFC83E94AA3CB13  TIME OF DISPATCH TO
                                           CPU 1 DISPATCHER/
                                           SCHEDULER ROUTINE
     0000001C  USERFLGC  A1                LOCK FLAGS
               USERLOC2 1..0  .00.         DISPATCH STATUS FLAGS FOR
                                           THE PRIMARY CPU DISPATCHER
```

Figure 13. Sample Output of the BLOCK Subcommand with the BITS and OFFSET Keywords

CHAIN Subcommand



Notes:

[1] Do not put blanks between the operands and special characters.

Purpose

Use the CHAIN subcommand to do any of the following:

- Display the addresses for the control blocks on a chain
- Display the data in the control blocks on a chain
- Display a count of the number of control blocks on a chain
- Detect any loops in a chain of control blocks.

This subcommand accepts a 24- or 31-bit qualifier for the address.

Operands

address
is a 1- to 8-digit hexadecimal address specifying the starting address of the first control block on the chain.

linkdisp
is a 1- to 6-digit hexadecimal operand specifying the displacement into the current control block where a pointer to the next control block in the chain is located. The valid range of the *linkdisp* operand is from hexadecimal 0 to FFFFFF.

?
specifies a 31-bit indirect address. A word (4 bytes) of storage at the specified address is read from the dump. This operand is the default.

%
specifies a 24-bit indirect address. A word (4 bytes) of storage at the specified address is read from the dump.

endval
is a 1- to 8-digit hexadecimal operand specifying the value of the pointer in the last block of the chain.

LIST
directs that only a list of control block addresses and a decimal count of the control blocks be displayed. The control blocks themselves are not to be displayed. LIST is the default for the command.

offset
: is a 1- to 6-digit hexadecimal offset indicating the starting address from which 4 contiguous bytes of data are to be displayed. The valid range of the offset operand is from X'0' to X'FFFFFF'. The default is 0.

COUNT
: is a keyword specifying that only a count of the total number of control blocks on the chain and not the addresses or control blocks themselves be displayed.

increment
: is a 1- to 4-digit decimal number designating how often the following message is issued: “nnnn ENTRIES - PROCESSING CONTINUES.” The default is 500 entries.

length
: is a 1- to 4-digit hexadecimal operand indicating the number of bytes to be displayed. The valid range of the length operand is from X'0' to X'1000'. However, the control block itself may be larger.

Usage Notes

1. The default indirect addressing mode is ? (question mark), which is 31-bit addressing mode. You can override the default addressing by specifying a percent sign (%) for 24-bit addressing mode.
2. If the number of control blocks on the chain exceeds 4096, a message is issued. Entering a null line continues the chain presentation starting with the last address displayed.
3. If you restart chain processing with the null line, the last address displayed becomes the first on the new chain.
4. If the address of the next control block in the chain has already been found in the current group of 4096 blocks, an error message is issued and processing ends.
5. If a loop of greater than 4096 entries exists, it is not detected.

Examples

Figure 14 on page 70 shows an example of a chain of control blocks.

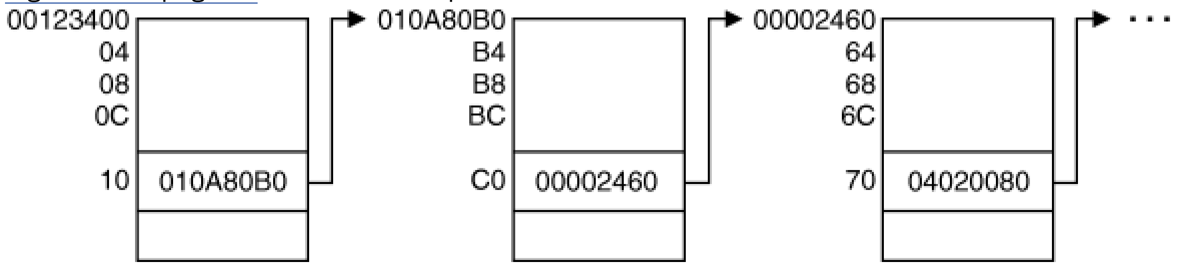


Figure 14. Example of a Chain of Control Blocks

Assume you know the following about the chain of control blocks:

- All the blocks in the chain have the same format.
- The address of the first block is 00123400.
- The pointer to the next block in the chain is at offset X'10' into the block.
- The last block in the chain contains a pointer value of zero.

If you wanted to view the addresses of the control blocks, you would enter this subcommand:

```
chain 123400 10 0
```

The output displayed appears in a list format:

```
CHAIN 123400 10 0
CB # 0001 AT 00123400
CB # 0002 AT 010A80B0
CB # 0003 AT 00002460
CB # 0004 AT 04020080
```

```
0004 ENTRIES WERE FOUND IN THE CHAIN
```

If you wanted to view the addresses of a chain of control blocks, but with 24-bit addressing specified, you would enter this command:

```
chain 203010 4% 0
```

The output displayed appears in a list format:

```
CHAIN 203010 4% 0
CB # 0001 AT 00203010
CB # 0002 AT 00203330
CB # 0003 AT 00203120
CB # 0004 AT 00203110
CB # 0005 AT 00203100
CB # 0006 AT 002030F0
CB # 0007 AT 002030E0
CB # 0008 AT 002030D0
       .
       .
```

The list would continue until all the blocks on the chain are listed. As with the example for 31-bit addressing, the total number of blocks on the chain are listed for 24-bit addressing requests. Figure 15 on page 71 shows an example of the output from the CHAIN subcommand when the length operand is specified.

The subcommand entered is:

```
chain 900818 0 0 50
```

```
CB  #(0001) ADDR(00900818?) LINKDISP(00000000) ENDV(00000000) LEN(00000050)
      0000  01F2C008  00000008  00000000  00000000 06 *.2..............*
      0010  00000000  00000000  00000000  00000000    *................*
      0020  00000000  00000000  00000000  00000000    *................*
      0030  00000000  00000000  C5D9C5D7  40404040    *........EREP    *
      0040  00000000  00000000  00000000  00000000    *................*

CB  #(0002) ADDR(01F2C008?) LINKDISP(00000000) ENDV(00000000) LEN(00000050)
      0000  00000000  00000000  E2E8E2E3  C5D44040 06 *........SYSTEM  *
      0010  01C47650  00000028  00000000  00000000    *.D.&;...........*
      0020  00000000  00000000  00000000  00000000    *................*
      0030  4CE5D4C3  4C4C4C4C  80000458  00E5D4C3    *<VMC<<<<.....VMC*
      0040  00000006  6E6E6E6E  00000000  00000000    *....>>>>........*

0002 ENTRIES WERE FOUND IN THE CHAIN
```

Figure 15. Sample Output of a CHAIN Subcommand with a Length Specified

Figure 16 on page 72 shows an example of the output from the CHAIN subcommand when LIST and an offset are specified. The subcommand entered is:

```
chain 362b000 600 0 list 540
```

```
CB # 0001 AT 0362B000 DATA => 00010000
CB # 0002 AT 03622000 DATA => 00010000
CB # 0003 AT 03615000 DATA => 00010000
CB # 0004 AT 03602000 DATA => 00010000
CB # 0005 AT 035F9000 DATA => 00010000
CB # 0006 AT 035F0000 DATA => 00010000
CB # 0007 AT 035E7000 DATA => 00020000
CB # 0008 AT 035B8000 DATA => 00010000
CB # 0009 AT 035AF000 DATA => 00010000
CB # 0010 AT 035A6000 DATA => 00010000
CB # 0011 AT 0359C000 DATA => 00010000
CB # 0012 AT 03767000 DATA => 00020000
CB # 0013 AT 0375A000 DATA => 00010000
CB # 0014 AT 03E2A000 DATA => 00010000

0014 ENTRIES WERE FOUND IN THE CHAIN
```

Figure 16. Sample Output of a CHAIN Subcommand with LIST and an Offset Specified

Responses

The hexadecimal address of each block found in the chain and a decimal count of the number of blocks found is displayed.

CMS Subcommand

```
►► CMS ──┬─────────┬──►◄
         └─command─┘
```

Purpose

Use the CMS subcommand to enter the CMS subset environment.

Operands

command
is any valid CMS command.

Usage Notes

1. If you enter the CMS subcommand without an operand, you enter CMS subset mode.
2. If you try to execute a CMS command that terminates abnormally, changes during the dump viewing session can be lost. You should try to save the current session file before using the CMS subcommand.
3. Any CMS command should be prefaced with CMS to prevent the Dump Viewing Facility or XEDIT from decoding the subcommand. This should be done to prevent cases where a CMS command can be interpreted as a Dump Viewing Facility or XEDIT subcommand.

CMSPOINT Subcommand (CMS Dump)

```
►►— CMSPoint —►◄
```

Purpose

Use the CMSPOINT subcommand to display the formatted contents of pointers from CMS NUCON.

Examples

Figure 17 on page 74 is an example of the output of the CMSPOINT subcommand.

```
LASTCMND= BEGIN 0
PREVCMND= FILELIST
LASTEXEC= PF
PREVEXEC= PROFILE
CURRSAVE= 0000C748
PGMSECT = 00002600
IOSECT  = 00002570
EXTSECT = 000024A0
ADTSECT = 000015F0
DEVTAB  = 00001390
DIOSECT = 00002940
SVCSECT = 000026A0
TAXEADDR= 000036B0
ALDRTBLS= 00100000
PGMOPSW = 00000001 60000002
PSAVE R0-R3=   00000000 00000000 00000000 00000000
PSAVE R4-R7=   00000000 00000000 00000000 00000848
PSAVE R8-R11=  000002A2 00000000 00000000 00000000
PSAVE R12-R15= 00020101 00000000 00020000 0001FA34
```

Figure 17. Example Output of the CMSPOINT Subcommand

Responses

CMSPOINT displays the formatted contents of the following CMS NUCON pointers:

Pointer	Formatted Contents
LASTCMND	Last command executed
PREVCMND	Previous command executed
LASTEXEC	Last exec executed
PREVEXEC	Previous exec executed
CURRSAVE	Address of the current system SVC save area
PGMSECT	Address of the program interrupt save area
IOSECT	Address of the I/O interrupt save area
EXTSECT	Address of the external interrupt save area
ADTSECT	Address of the first active disk table
DEVTAB	Address of the CMS device table
DIOSECT	Address of the disk I/O work area
SVCSECT	Address of the SVC handler control block used by DMSITS
TAXEADDR	Address of the terminal attention interrupt exit

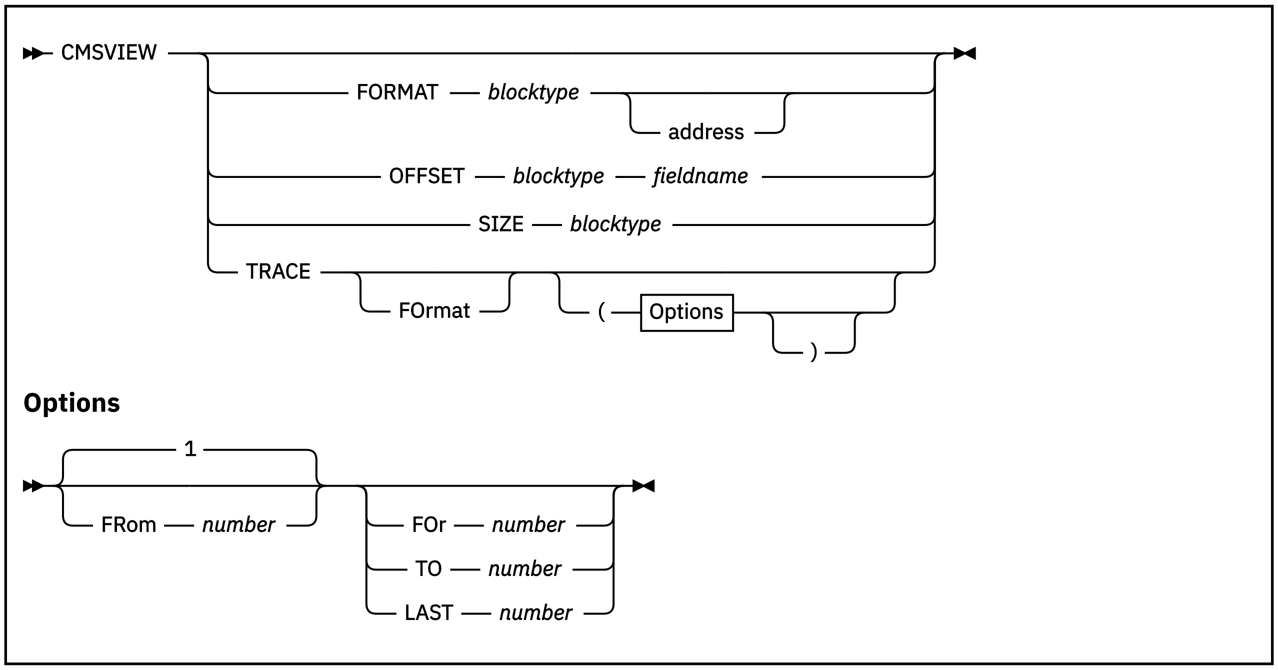
Pointer	Formatted Contents
ALDRTBLS	Address of the loader tables
PGMOPSW	Program old PSW
PSAVE	Contents of 16 general registers at time of abend from PGMSECT

Messages and Return Codes

DMSDDP2017I
PAGE '*page*' NOT FOUND IN DUMP

DMSDFR2017E
INVALID OPERAND - *operand*

CMSVIEW Subcommand



Purpose

Use the CMSVIEW subcommand (macro) to format CMS control blocks, traverse CMS control block chains, and display CMS trace data.

Operands

FORMAT *blocktype*
: formats a CMS control block and displays it on your screen. If the block is a one-of-a-kind block (for example, the KGA), you need not supply an address.

OFFSET *blocktype fieldname*
: shows you the offset within a block to a given field.

SIZE *blocktype*
: shows you the size of a block.

TRACE
: shows you the accumulated trace data.

  FOrmat
  : indicates that the TRACE subcommand output should be formatted before being displayed. If the FORMAT option is not specified, the output is not formatted before being displayed. See "CMSVIEW TRACE Subcommand" on page 81 for more information.

Options

FROM *number*
: specifies the first trace entry to display. *number* is an integer trace entry number. The default is 1.

FOR *number*
: specifies how many trace entries are to be displayed, where *number* is an integer from 1 to the number of entries in the trace table. The default is to display all trace entries starting with the FROM value.

TO ***number***
: specifies the last trace entry to be displayed, where *number* is an integer from 1 to the number of entries in the trace table. The default is to display all trace entries starting with the FROM value.

LAST ***number***
: specifies how many trace table entries are to be displayed starting from the end or most recent ("last") entries, where *number* is an integer from 1 to the number of entries in the trace table. LAST overrides all other specified operands.

Usage Notes

1. Dumps to be analyzed by CMSVIEW should be taken with the `VMDUMP 0-END FORMAT CMS` command.
2. If you specify no command when you start CMSVIEW, a window will appear on the screen and you will be asked to enter a CMSVIEW command.
3. When CMSVIEW draws a formatted control block, the PF keys are set to allow certain convenient functions:

   Note: Switch among PF key sets with PF10.

   PF Keys (Set 1)

   PF1
   Displays the HELP information for CMSVIEW

   PF2
   Formats the block indicated by the cursor position

   PF3
   Return to previous CMSVIEW window

   PF4
   Displays the KGA

   PF5
   Displays the PLD for CPU 0

   PF6
   Displays the TSD of the thread running on CPU 0

   PF7
   Scrolls backward through the displayed window

   PF8
   Scrolls forward through the displayed window

   PF9
   Displays storage pointed to by contents at cursor

   PF10
   Swaps to the next PF key set

   PF11
   Displays the PSD of the root process

   PF12
   Displays the PSD of the commands process

   PF Keys (Set 2)

   PF1
   Display a prompt for a CMSVIEW command

   PF2
   Display a prompt for a CMS command

   PF3
   Return to previous CMSVIEW window

PF4

PF5

PF6

PF7
Scrolls backward through the displayed window

PF8
Scrolls forward through the displayed window

PF9

PF10
Swaps to the next PF key set

PF11

PF12

PF Keys (Set 3)

PF1

PF2

PF3
Return to previous CMSVIEW window

PF4

PF5

PF6

PF7
Scrolls backward through the displayed window

PF8
Scrolls forward through the displayed window

PF9

PF10
Swaps to the next PF key set

PF11

PF12

Examples

1. The output received for `CMSVIEW FORMAT KGA` might look like this (where the *v*, *r*, and *m* displayed will be the version, release, and modification of your z/VM system):

```
  z/VM VvRr.m - Dumpscan   DUMP DUMP0001 F1    Type=VM Format=CMS
 HCSDSS200I PROCESSING FILE DUMP DUMP0001 F1
H + ------------------------------------------------------------ +
* |  In KGA at address 005DD000 (length X'6B0'), you find...      |
  |                                                              |
  |  Offset  Field_Name               What       Contents         |
  |  ------  ----------               ----       --------         |
  |  0000    kga_abn_anch             *KABNE     005DD6B0         |
  |  0004    kga_abn_len                         00000010         |
  |  0008    kga_act_anch             *KACTE     005DD6C0         |
  |  000C    kga_act_len                         00000020         |
  |  0010    kga_cpu_anch             *KCPUE     005DD6E0         |
  |  0014    kga_cpu_len                         00000228         |
  |  0018    kga_evn_anch             *KEVNE     005DD908         |
  |  001C    kga_evn_len                         00000050         |
  |  0020    kga_ipc_anch             *KIPCE     005DD958         |
  |  0024    kga_ipc_len                         000003E8         |
  |                                                              |
  |  PF1=Help   2=ToBlk  3=Quit   4=KGA    5=PLD.0  6=TSD.0       |
  |  PF7=Bkwd   8=Fwd    9=AsStg 10=SwPF  11=Root  12=Cmds        |
  + ------------------------------------------------------------ +

 ====> cmsview format kga
```

2. The output received for `CMSVIEW OFFSET KGA KGA_ACT_LEN` might look like this:

```
Offset within KGA to kga_act_len is X'C'.
```

3. The output received for `CMSVIEW SIZE KGA` might look like this:

```
Size of KGA is X'6B0' bytes.
```

Messages and Return Codes

The following messages may be returned by the CMSVIEW subcommand:

DMSSB$002E
File *fn ft* not found

DMSSB$2551E
No block to format specified

DMSSB$2552E
No block name to search specified

DMSSB$2553E
No field to find specified

DMSSB$2554E
No block to size specified

DMSSB$2555E
Address of *block* could not be determined

DMSSB$2556E
Address *addr* is not a likely place to find a *block*

DMSSB$2557E
Offset of *field* within *block* could not be determined

DMSSB$2558I
Offset within *block* to *field* is *hexdisp*

DMSSB$2559E
Size of *block* could not be determined

DMSSB$2560E
Size of *block* is X'*xx*' bytes

DMSSB$2561E
There is a *block* at address *addr*, but its format is not known. *filename* BLOCKDEF file is probably incorrect

DMSSB$2562E
Error *returncode* loading *filename* BLOCKDEF *

DMSSB$2564E
Error on READSTRG command (rc=*rc*). Trace data processing stopped (address=*vaddr*)

DMSSB$2565E
No trace data could be found

DMSSB$2566E
Unknown error

DMSSB$2567W
FOR number is not a positive whole number. Set to *n*

DMSSB$2567W
FROM number is not a positive whole number. Set to 1

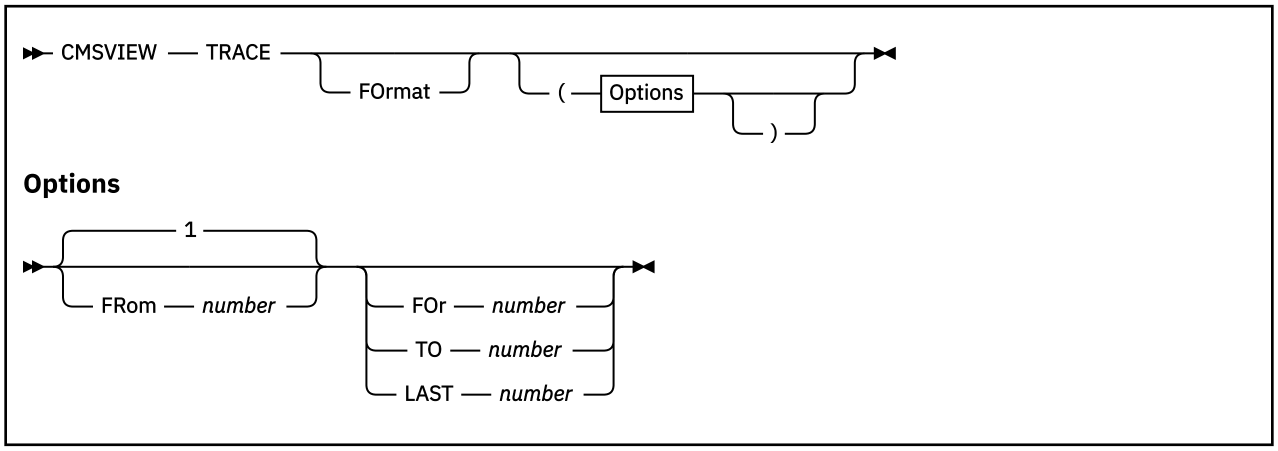
DMSSB$2567W
FROM number too big. Set to 1

DMSSB$2567W
LAST number is not a positive whole number. Set to *n*

DMSSB$2567W
TO number is not a positive whole number. Set to *n*

DMSSB$2567W
TO/FOR value too big. Set to *n*

CMSVIEW TRACE Subcommand



Purpose

Use the CMSVIEW TRACE subcommand to display the CMS trace data.

Operands

FOrmat
: indicates that the TRACE subcommand output should be formatted before being displayed. If the FORMAT option is not specified, the output is not formatted before being displayed.

Options

FROM *number*
: specifies the first trace entry to display. *number* is an integer trace entry number. The default is 1.

FOR *number*
: specifies how many trace entries are to be displayed, where *number* is an integer from 1 to the number of entries in the trace table. The default is to display all trace entries starting with the FROM value.

TO *number*
: specifies the last trace entry to be displayed, where *number* is an integer from 1 to the number of entries in the trace table. The default is to display all trace entries starting with the FROM value.

LAST *number*
: specifies how many trace table entries are to be displayed starting from the end or most recent ("last") entries, where *number* is an integer from 1 to the number of entries in the trace table. LAST overrides all other specified operands.

Usage Notes

1. The trace table is not a reserved area of storage, but a set of signals of the Trace event. The trace table cannot be examined effectively without the help of the CMSVIEW TRACE subcommand.
2. Each trace entry is identified by a number that orders the entries in time, trace entry number one being the oldest in the trace table. Trace entries are displayed "youngest" to "oldest".
3. The trace code is given in two parts: the first identifies the type of trace event and the second gives the more specific subtype. For trace entry 99 in the example, "02 02" indicates a dispatching event (02) which is specifically a subtype of 02, or *promote thread* event.

Examples

1. The output received for `CMSVIEW TRACE FORMAT ( FROM 99 TO 100` might look like this:

```
Num    CPU  TOD Clock         Code    Account-id  Pid  Tid
100    00   A467257D4405CE03  02 002  CLIENTU4     02   01
       Promote by 00442A50 dcd 004421A0 chose 00442A50 called by 80325394

Num    CPU  TOD Clock         Code    Account-id  Pid  Tid
99     00   A467257D43FF6503  02 004  CLIENTU4     02   01
       Sched by 00442A50 who 002F4320 called by 80327862
```

2. The output received for `CMSVIEW TRACE ( FROM 99 TO 100` might look like this:

```
100    00000044  00000002  00000002  00000000
       00000000  00000000  00000000  E2C8E4D3
       E3E9F240  00000002  00000001  00000000
       A467257D  4405CE03  40404040  40404040
       00000010  00442A50  00442A50  004421A0
       80325394

99     00000044  00000002  00000004  00000000
       00000000  00000000  00000000  E2C8E4D3
       E3E9F240  00000002  00000001  00000000
       A467257D  43FF6503  40404040  40404040
       0000000C  00442A50  002F4320  80327862
```

Messages and Return Codes

The following messages may be returned by the CMSVIEW TRACE subcommand:

DMSSB$002E
File *fn ft* not found

DMSSB$2564E
Error on READSTRG command (rc=*rc*). Trace data processing stopped (address=*vaddr*)

DMSSB$2565E
No trace data could be found

DMSSB$2566E
Unknown error

DMSSB$2567W
FOR number is not a positive whole number. Set to *n*

DMSSB$2567W
FROM number is not a positive whole number. Set to 1

DMSSB$2567W
FROM number too big. Set to 1

DMSSB$2567W
LAST number is not a positive whole number. Set to *n*

DMSSB$2567W
TO number is not a positive whole number. Set to *n*

DMSSB$2567W
TO/FOR value too big. Set to *n*

CPU Subcommand

```
►► CPU ►◄
```

Purpose

Use this subcommand to display the CPU address and the prefix register value for each processor in the dump.

Usage Notes

1. This subcommand does not clear the screen before displaying information.
2. The failing processor is always listed first.
3. For more information about any processor in the system, use the REGS, CREGS, or GREGS subcommands.

Examples

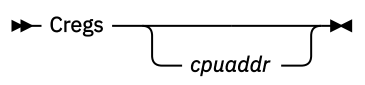
Figure 18 on page 83 illustrates the output you receive after entering the CPU subcommand. The subcommand entered is:

```
cpu
```

```
CPU ADDRESS IS 0000    PREFIX REGISTER IS 0095E000   (FAILING)
CPU ADDRESS IS 0001    PREFIX REGISTER IS 01CB0000
```

Figure 18. Sample Output of a CPU Subcommand

CREGS Subcommand



Purpose

Use the CREGS subcommand to display the control registers for a specified CPU address.

Operands

cpuaddr
is a 1- to 4-digit hexadecimal number specifying the processor address for which the information is to be displayed.

Usage Notes

1. If the *cpuaddr* operand is not specified, it defaults to processor on which the CP VMDUMP command was issued for the virtual machine.
2. Use the CPU subcommand to obtain the processor addresses in the dump.

Examples

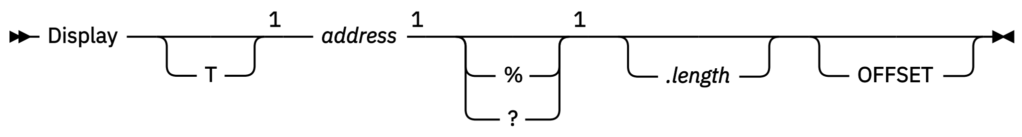
Figure 19 on page 84 illustrates the output of the CREGS subcommand for the failing processor. The subcommand entered is:

```
Cregs
```

```
CPU ADDRESS - 0000
CONTROL REGS  0 - 15
     90B0FE40 00800001 00000000 00000000  00000000 00000000 80000000 00000000
     00000000 00000000 00000000 00000000  01FA8681 00000000 5F000000 00000000
```

Figure 19. Sample Output of a CREGS Subcommand

DISPLAY Subcommand



Notes:

[1] Do not put blanks between the operands and special characters.

Purpose

Use the DISPLAY subcommand to display areas of the dump. You can specify an address, an indirect address, or request data to be displayed by offsets from an address.

Operands

address
: is the 31-bit (4-byte) hexadecimal address from which the data is to be displayed.

%
: specifies a 24-bit indirect address. A word (4 bytes) of storage at the specified address is read from the dump. The low-order 24 bits are used to compute the address that is displayed.

?
: specifies a 31-bit indirect address. A word (4 bytes) of storage at the specified address is read in from the dump. The low-order 31 bits are used to compute the address displayed.

.length
: is an optional operand. It is a 1- to 4-digit nonzero hexadecimal number indicating the length in bytes to be displayed. The valid range is from X'1' to X'FFFF'. If this is specified, the screen is **not** cleared. If it is not specified, the screen is cleared and the output is displayed.

One screen of dump data is presented in both hexadecimal and EBCDIC.

OFFSET
: is an optional operand. If you specify it, the leftmost column of the output contains the offsets from the input address instead of the storage address of the data. The data is displayed to the right of the column of offsets.

Usage Notes

1. A period (.) used as a delimiter between the address and the length is acceptable. If the indirect addressing qualifier is specified, the delimiter should follow the qualifier.
2. A *T* preceding the *address* operand (for example, T*address*) is used to provide compatibility with the CP DISPLAY command. The DISPLAY subcommand always provides the EBCDIC translation whether *T* is specified or not.
3. If you specified an indirect address, the resulting address appears in parentheses in the output as part of the command line.
4. The minimum output is one 16-byte line with EBCDIC translation.
5. If you specify the *length* operand, the resulting display is rounded to start and end on a 10-byte (hexadecimal) boundary (see Figure 23 on page 87).
6. Each line includes 16 bytes of hexadecimal data with the EBCDIC translation. In addition, the key of the page is displayed on the first line and subsequent page boundaries. It appears between the hexadecimal data and its EBCDIC translation.
7. OFFSET and *length* are operands that may be specified in any order.

8. If you specify the OFFSET keyword operand, the leftmost column of the output contains the hexadecimal offset from the starting address instead of the 31-bit storage address.
9. If only partial data is available in the dump, DISPLAY presents all of the available data, and then an error message is displayed.

Examples

Figure 20 on page 86 illustrates the results of the command DISPLAY 11A9D0. Notice that the left-most column contains the storage address of the data. The subcommand entered is:

```
Display 11a9f0
```

```
DISPLAY 11A9F0
  0011A9F0  F14BF4F7  D9C50000  05C04BC0  C0064700    *1.47RE..........*
  0011AA00  002A41C0  C0009180  728A47E0  C29A58E0    *......j.....B...*
  0011AA10  868818FE  89F00001  58100974  88100002    *fh..i0......h...*
  0011AA20  18D15BD0  07508DE0  100089E0  0001185E    *.J$..&;...i....;*
  0011AA30  41400005  186407F0  9514089A  4770C0AE    *. .....0n.......*
  0011AA40  58F0B520  12FF4770  C2085820  C2381733    *.0......B...B...*
  0011AA50  18134333  20031233  4780C0AE  58A32000    *.............t..*
  0011AA60  12AA47B0  C0AE54A0  07B45890  A60C4190    *............w...*
  0011AA70  96589101  90034780  C21847F0  C0805820    *o.j.....B..0....*
  0011AA80  08049180  20004710  C2081733  18134333    *..j.....B.......*
  0011AA90  20031233  4780C0EA  58A32000  12AA47B0    *.........t......*
  0011AAA0  C0EA54A0  07B45890  A60C4190  96589101    *........w...o.j.*
  0011AAB0  90034780  C21847F0  C0BC4110  00144610    *....B..0........*
  0011AAC0  C0EE91C0  08024770  C2924660  C0664940    *..j.....Bk.-... *
  0011AAD0  C2504740  C1164940  C25247B0  C11A4140    *B&; A.. B...A.. *
  0011AAE0  400247F0  C11A8940  00011864  951E089A    * ..0A.i ....n...*
  0011AAF0  4780C1CE  41202080  41DD0001  5520C23C    *..A...........B.*
  0011AB00  4770C13A  5820C240  17DD8950  00011255    *..A...B ..i&;...*
  0011AB10  4780C1CE  47B0C124  45E02004  95002003    *..A...A.....n...*
  0011AB20  4780C124  17331813  43332003  12334780    *..A.............*
  0011AB30  C12458A3  200012AA  47B0C124  54A007B4    *A..t......A.....*
  0011AB40  58F0A530  12FF4770  C1565890  A60C4190    *.0v.....A...w...*
  0011AB50  96589101  90034770  C1561892  1E939108    *o.j.....A..k.lj.*
  0011AB60  90024710  C1565890  A5205690  A5245490    *....A...v...v...*
  0011AB70  C2444770  C20E5890  A5285490  A52C4770    *B...B...v...v...*
  0011AB80  C1565890  C2485E90  09745890  90005490    *A...B.;.........*
  0011AB90  A5285590  A5284770  C15647F0  C20E17FF    *v...v...A..0B...*
  0011ABA0  BFF80832  57F00748  54F00834  17114120    *.8...0...0......*
  0011ABB0  09D012FF  4720C1F4  4780C066  B23F0000    *......A4........*
  0011ABC0  4740C200  89F00001  41101001  47F0C1E2    *. B.i0........0AS*
  0011ABD0  41F00008  47F0C21A  17FF47F0  C22A186D    *.0...0B....0B...*
  0011ABE0  41F00004  47F0C21A  17FF58E0  C24C58E0    *.0...0B.....B<..*
  0011ABF0  E00056E0  86504770  C2548200  C2300000    *....f&;.B.b.B...*
  0011AC00  000C0000  800638A6  00063A80  00064A80    *.......w........*
```

Figure 20. Sample Output of a DISPLAY Subcommand without the Length Operand

Figure 21 on page 86 illustrates the display generated using 31-bit indirect addressing with a length. The subcommand entered is:

```
display 9f0080? 40
```

```
DISPLAY 9F0080  ?  40   (00800000)
  00800000  80000020  80000020  80000020  80000020 06 *................*
  00800010  80000020  80000020  80000020  80000020    *................*
  00800020  80000020  80000020  80000020  80000020    *................*
  00800030  80000020  80000020  80000020  80000020    *................*
```

Figure 21. Sample Output of a DISPLAY Subcommand with 31-Bit Indirect Addressing

Figure 22 on page 87 illustrates the DISPLAY command using the OFFSET operand. Note that the left-hand column is the offset from the address entered. The subcommand entered is:

```
d fca5f4? offset
```

```
DISPLAY FCA5F4  ?       (00FD9C80) OFFSET
      0000  00FD9D00  00000000  00000000  00026000 06 *..............-.*
      0010  00000000  000E85E2  E4C94040  00000041    *......eSUI  ....*
      0020  00001000  00F88238  00121028  00000000    *.....8b.........*
      0030  00001000  00FDA700  00014000  00000000    *......x... .....*
      0040  00F88238  00001000  0005FF30  00000018    *.8b.............*
      0050  8006003C  00121028  00000000  00000000    *................*
      0060  00000000  00000000  00000000  00000000    *................*
      0070  00000000  00000000  00000000  00000000    *................*
      0080  00FD9400  00000000  00000000  00026000    *..m...........-.*
      0090  00000000  000E85E2  00E4E1C0  00FDA9D8    *......eS.U....zQ*
      00A0  00000001  00000000  A1C186C7  8BD46412    *.........AfG.M..*
      00B0  00000000  00FDAAE0  00FDA700  80D41AA8    *..........x..M.y*
      00C0  00FDAAE0  00001000  00D41000  00FD9C80    *.........M......*
      00D0  80D41BAC  000EFCE8  00000000  00000000    *.M.....Y........*
      00E0  00000000  00000000  00000000  00000000    *................*
      00F0  00000000  00000000  00000000  00000000    *................*
      0100  00FD9380  00000000  00000000  00026000    *..l...........-.*
      0110  00000000  000E85E2  A1C18738  00000000    *......eS.Ag.....*
      0120  000F1274  000F0FB8  000F11E0  00A7E030    *.............x..*
      0130  80082C74  000F1274  00016B80  0007DDA0    *..........,.....*
      0140  00FD85F8  00001000  00082C20  00FD9400    *..e8..........m.*
      0150  80082EA6  000FFBF8  00000000  00000000    *...w...8........*
      0160  00000000  00000000  00000000  00000000    *................*
      0170  00000000  00000000  00000000  00000000    *................*
      0180  00000000  00000000  00000000  00026000    *..............-.*
      0190  00800000  000E8348  00000000  0C500020    *......c......&;.*
      01A0  42424048  0038F720  00003A10  00000000    *.. ...7.........*
      01B0  000F4F80  00000398  00016400  0007DDA0    *..|....q........*
      01C0  00FD8138  00001000  0007CDA0  00000038    *..a.............*
      01D0  8007D28C  0006E810  00000000  00000000    *..K...Y.........*
      01E0  00000000  00000000  00000000  00000000    *................*
      01F0  00000000  00000000  00000000  00000000    *................*
      0200  00000000  00000000  00000000  00026000    *..............-.*
      0210  00800000  000E85E2  0000000B  00000001    *......eS........*
      0220  00000014  00000003  00000000  00000000    *................*
      0230  00FD9268  00014000  00016400  00000000    *..k... .........*
```

Figure 22. Sample Output of a DISPLAY Subcommand with the OFFSET Operand

Figure 23 on page 87 illustrates the DISPLAY subcommand specified with a length. Note that the data displayed begins and ends on a 10-byte (hexadecimal) boundary. The subcommand entered is:

```
display 9efff6 c
```

```
DISPLAY 9EFFF6     C
  009EFFF0  5810D038  58001C24  18185410  0B0C8910 04 *................*
  009F0000  000A1E01  1FEE1FFF  1F225810  2C1CBB0E 06 *................*
```

Figure 23. Sample Output of a DISPLAY Subcommand with a Length Specified

DOSPOINT Subcommand (CMS Dump)

```
►► DOSPoint ►◄
```

Purpose

Use the DOSPOINT subcommand to display the formatted contents of five pointers used by DOS simulation.

Usage Notes

1. If the DOSPOINT subcommand is invoked and DOS simulation is not in effect, an error message is displayed.

Examples

```
DMSDDP2017I DOS SIMULATION NOT IN EFFECT
BGCOM   = 00000DB8
SYSCOM  = 00000CA0
LTASAVE= 00001180
ACBLIST= 00000000
DOSSECT= 00000000
```

Responses

DOSPOINT displays the formatted contents of the following DOS simulation pointers:

Pointer	Formatted Contents
BGCOM	Address of the background communications area
SYSCOM	Address of the systems communication area
LTASAVE	Address of the logical transient save area
ACBLIST	Address of the ACB list built by OPEN/CLOSE
DOSSECT	Address of the first DOSCB control block

Messages and Return Codes

DMSDDP2017I
DOS SIMULATION NOT IN EFFECT

DMSDDP2017I
Page '*page*' NOT FOUND IN DUMP

DMSDFR2017I
INVALID OPERAND - *operand*

DUMPID Subcommand

```
►► DUMPID ►◄
```

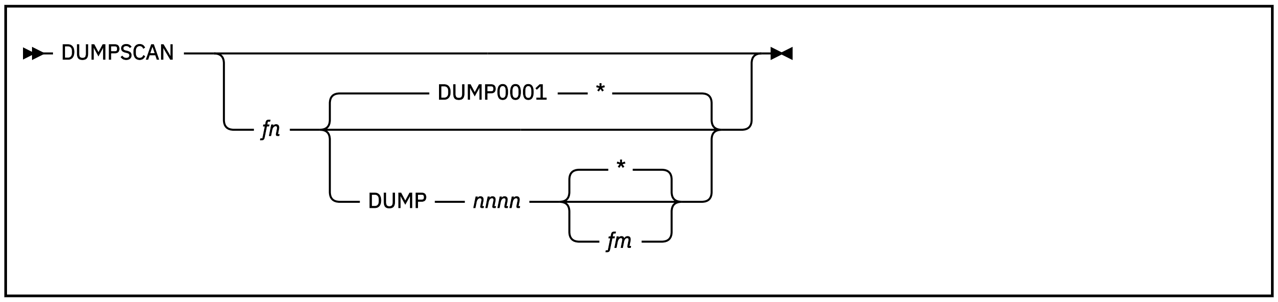
Purpose

Use the DUMPID subcommand to display the dump identifier.

Usage Notes

1. This subcommand is valid for virtual machine dumps. For VM dumps the identifier is assigned with the VMDUMP command. For more information on the VMDUMP command, see the *z/VM: CP Commands and Utilities Reference*.

DUMPSCAN Subcommand



Purpose

Use the DUMPSCAN subcommand to view a different dump file without leaving the DUMPSCAN command environment.

Operands

fn
: is the name of the dump file to be processed.

DUMP*nnnn*
: is the file type of the dump file, where *nnnn* is 4-digit number. The default file type is DUMP0001.

fm
: is the file mode of the dump file. If it is omitted, or is an asterisk, DUMPSCAN uses the standard CMS search sequence.

Usage Notes

1. The DUMPSCAN file has the same file name as the dump file. The file type is VIEW*nnnn* where *nnnn* is the number from the dump file type, DUMP*nnnn*. The new file is added to the current DUMPSCAN command file ring, or if the file is already in the file ring, then the DUMPSCAN subcommand switches to it and the current screen is updated to reflect the new dump file being viewed.
2. Because the DUMPSCAN command and subcommand share the same name, the user needs to know which DUMPSCAN was invoked in order to avoid possible confusion when viewing multiple dumps. See the Usage Guide section under "Viewing Several Dump Files at a Time" on page 11 for further information on viewing multiple dump files.
3. If the dump is on a writable disk, the session file uses the file mode of the dump file. Otherwise the file mode is "A".
4. If the DUMPSCAN subcommand is entered with no operands, it switches to the next VIEW*nnnn* file in the current DUMPSCAN command file ring. If there are no other VIEWnnnn files, DUMPSCAN remains at the current dump file.
5. The session file uses the file mode A.
6. The dump viewing session can be filed with the XEDIT FILE subcommand. When you view the same dump later, the saved file is reactivated and the new session is appended to the *dumpname* VIEW*nnnn* file containing the previous sessions.

Messages and Return Codes

Return Code	**Explanation**
0	Successful completion

4
: An error message has been issued

8
: Error trying to add a file to the ring

50
: CP dumps are not supported. Use VM Dump Tool

104
: Internal processing error

END Subcommand

```
►► END ►◄
```

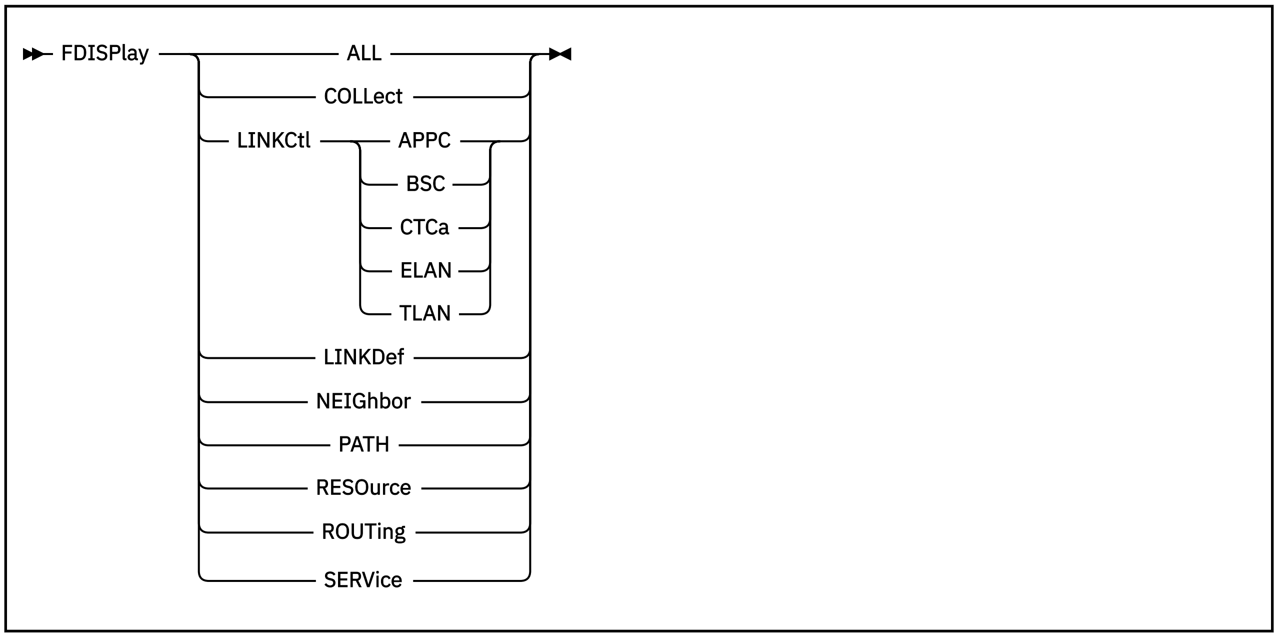
Purpose

Use the END subcommand to end the session.

Usage Notes

1. The END subcommand is equivalent to the XEDIT QQUIT subcommand.
2. If you enter the END subcommand and you have changed the session file, you must enter the XEDIT FILE subcommand to save the session file. If you do not wish to save the session file, enter the DUMPSCAN QUIT or DUMPSCAN HX subcommand.

FDISPLAY Subcommand (TSAF Dump)



Purpose

Use the FDISPLAY (Formatted Display) subcommand to display data control blocks, tables, and arrays important to the TSAF virtual machine.

Operands

ALL
: displays all of the information produced by the operands of this subcommand; this is useful when you want a hard copy of the information.

COLLect
: displays the collection control block.

LINKCtl
: displays the link control blocks for the following types of links:

  APPC
  : logical advanced program-to-program communications links.

  BSC
  : bisynchronous communications links.

  CTCa
  : channel-to-channel adapter links, which include channel-to-channel adapters and 3088 drivers.

  ELAN
  : IBM 9370 IEEE 802.3 local area network (LAN) subsystem links.

  TLAN
  : IBM 9370 Token Ring LAN subsystem links.

LINKDef
: displays the link definition array.

NEIGhbor
: displays the neighbor table.

PATH
: displays the path array.

RESOurce
: displays the resource table.

ROUTing
: displays the routing array.

SERVice
: displays the service table.

Usage Notes

1. To produce a SPOOL file with the data, enter the following commands (*nnn* is the number of trace entries; the maximum is 999):

```
    print fdisplay all
    print trace format for nnn
    print close
```

Examples

Figure 24 on page 94, Figure 25 on page 94, Figure 26 on page 94, and Figure 27 on page 94 are examples of the output of the FDISPLAY subcommand.

```
 ***  TSAF Link-Definition Table ***

Entry number: 1
Symdest Name: APPCLINK                 ATSLINKS FILE record number: 1
Number of bytes read: 365232           Number of bytes sent: 448136
Time link came up: 0                   Link delay: 54
Driver index: 6                        Link control block address: 0099A828
Link state: 1                          Link flags: 'A02000'X

Read/Write units: 03E0/03E0            ATSLINKS FILE record number: 4
Number of bytes read: 16560            Number of bytes sent: 18040
Time link came up: 0                   Link delay: 154
Driver index: 1                        Link control block address: 003F9D40
Link state: 1                          Link flags: 'A00000'X
```

Figure 24. Sample Output of the FDISPLAY Subcommand (LINKDEF Operand)

```
 ***  BiSync Link Control Blocks  ***

 ATSZBD096I THE DATA STRUCTURE IS EMPTY
```

Figure 25. Sample Output of the FDISPLAY Subcommand (LINKCTL BSC Operand)

```
 ***  CTC/3088 Link Control Blocks  ***

Write CCW:  01032F90 20000004          Read CCW:  022C0408 200001F8
Sense CCW:  043F9D65 20000001          Link number:  1
Send queue front pointer:  00000000    Send queue rear pointer: 00000000
Link state: 0                          Sense byte: '40'X
```

Figure 26. Sample Output of the FDISPLAY Subcommand (LINKCTL CTCA Operand)

```
 ***  TSAF Routing Table  ***

Destination processor:  GDLVMA         Via link number:   2
Entry flags:  '80000000'X              Path version number:  9
Link weight:  50                       Hop count:  0

Destination processor:  VMB            Via link number:   1
Entry flags:  '80000000'X              Path version number:  9
Link weight:  50                       Hop count:  0
```

Figure 27. Sample Output of the FDISPLAY Subcommand (ROUTING Operand)

Messages and Return Codes

ATSZTD078E
OPERAND MISSING OR INVALID

ATSZTD084I
PAGE '*page*' NOT FOUND IN DUMP

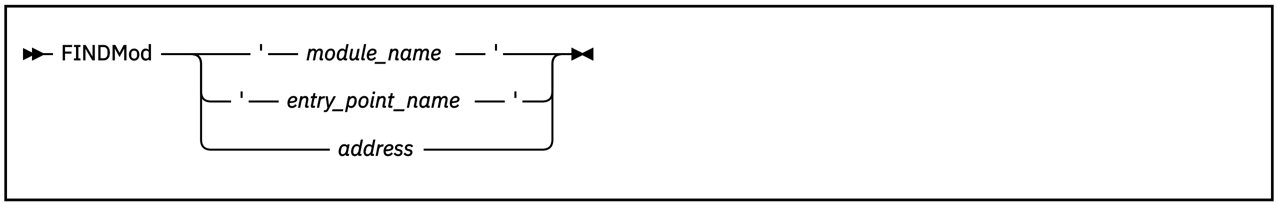
ATSZTD089E
UNABLE TO LOCATE GLOBAL CONTROL BLOCK (ATSCGM)

ATSZ5D094I
THE POINTER TO THE SPECIFIED STRUCTURE IS ZERO

ATSZ5D095I
THE LINK WAS NOT FOUND IN THE LINK-TYPE TABLE

ATSZ5D096I
THE DATA STRUCTURE IS EMPTY

FINDMOD Subcommand



Purpose

Use this subcommand either to locate a specified module or entry point in the dump or to locate the module and the entry point that resides at a specified address.

Operands

module_name
: is an alphanumeric string of 1 to 8 characters that specifies a module name. Place this variable within single quotation marks.

entry_point_name
: is an alphanumeric string of 1 to 8 characters that specifies an entry point name. Place this variable within single quotation marks.

address
: is a 31-bit (4-byte) hexadecimal address.

Usage Notes

1. The FINDMOD subcommand requires that a module map of the dump be appended to the dump.
2. If the *module name* or *entry point name* operands are entered, the starting address of the module or entry point is displayed on the first line of the screen.
3. If the requested module is not in storage, an error message is displayed.
4. A string in quotation marks is processed as a module or entry point name. A string not in quotation marks is handled first as an address. If the string is not a valid hexadecimal number, it is then processed as a module or entry point name. If the string is not a module and is not a valid hexadecimal address, an error message is displayed indicating the string is not a valid hexadecimal address.
5. The output displays the name and hexadecimal location of the next lowest entry point and the displacement of the address from that entry point address, as well as the name of the module containing that entry point and the displacement from its start.
6. Scrolling subcommands—BACKWARD, FORWARD, + (increment), - (decrement)—can be used after the module is found.
7. If the specified address is not in the dump, an error message is displayed.
8. If the operand entered is either a module name or an entry point name and it exits in the dump, the screen is cleared before the information is displayed.
9. If the operand entered is an address, the screen is not cleared prior to display of the information.

Examples

Figure 28 on page 97 illustrates the data displayed when the FINDMOD subcommand is issued to locate module DMSWMI. The subcommand entered is:

```
findmod 'DMSWMI'
```

```
01102600  47F0F066  00BC10C4  D4E2E6D4  C9404040    *.00....DMSWMI   *
01102610  40F9F74B  F3F4F700  47F0F04E  00A447F0    * 97.347..00+.u.0*
01102620  F04801EE  47F0F042  026047F0  F03C02D2    *0....00..-.00..K*
01102630  47F0F036  036E47F0  F030040A  47F0F02A    *.00..>.00....00.*
01102640  046C47F0  F02405DA  47F0F01E  079E47F0    *.%.00....00....0*
01102650  F0180864  47F0F012  08F447F0  F00C0984    *0....00..4.00..d*
01102660  47F0F006  0A4A90EC  D00C05C0  5800CE24    *.00.............*
01102670  58F00DE0  5810F040  1E015500  F04447D0    *.0....0 ....0...*
01102680  C0361F01  58F00DDC  05EF0000  00400000    *.....0....... ..*
01102690  00000000  00080000  000458E0  D00C58F0    *...............0*
011026A0  0DE05000  F04018FD  18D150F0  D00450D0    *..&;0 ...J&X-0;.&;*
011026B0  F00898F1  F0104AFF  000407FF  5860913C    *0.q10........-j.*
011026C0  1F889110  101D47E0  C1621F77  4370101C    *.hj.....A.......*
011026D0  41B00001  197B4740  C15A4780  C11241F0    *.....#. A...A..0*
011026E0  0004197F  4720C15A  4780C0D2  06708970    *..."..A....K..i.*
011026F0  000247F7  C08647F0  C13647F0  C09245E0    *...7.f.0A..0.k..*
```

Figure 28. FINDMOD Output When a Module Name Is Specified

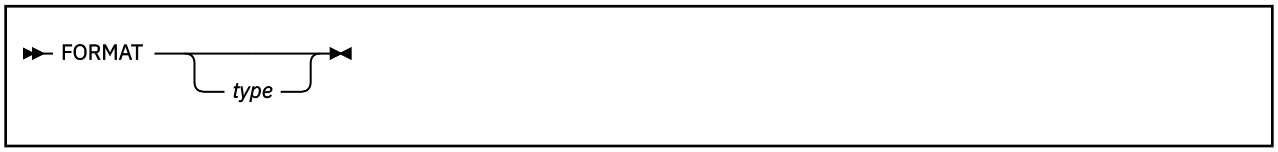
Figure 29 on page 97 illustrates the data displayed when an address operand (5ad20) is specified to determine the module that has the address. The subcommand entered is:

```
findmod 01102680
```

```
01102680 is 80 bytes into module DMSWAI at 01102600
```

Figure 29. FINDMOD Output When an Address Is Specified

FORMAT Subcommand



Purpose

Use the FORMAT subcommand to change or query the type of a virtual machine dump being viewed.

Operands

type
is any 1- to 8-character name for the new dump type.

Usage Notes

1. The type of a dump is not the same as the file type of a CMS dump file. The dump type identifies dumped data's format, such as can be specified with the FORMAT parameter of the CP VMDUMP command.
2. If *type* is not specified, FORMAT displays the current dump type on the user's screen.
3. If the format is changed, FORMAT displays the new type on the screen, appends it to the session file, and changes the DUMPSCAN status line.
4. If *type* is one of the supported dump types, FORMAT unloads or resets the block files and exit routines for the old dump type and loads the ones for the new dump type. For instance, if a dump of CMS data has been made with the default type of FILE, entering

   ```
   FORMAT cms
   ```

   changes the format of the dump and loads the CMS-only routines, such as CMSPOINT, for examining the dump.

   See Appendix A, "Using Attachment Interfaces," on page 139 for a discussion of the supported dump types and the exit routines for extracting and formatting.
5. Because information within the dump is modified as a result of this command, you must have write access to the dump file.
6. This subcommand is for virtual machine dumps only.

Examples

When

```
FORMAT SFS
```

is entered, the new format of the dump is displayed at the user's screen:

```
DUMP FORMAT IS: SFS
```

Messages and Return Codes

Return Code
Explanation

0
Successful completion

4
: The dump is on a disk that is not accessed as R/W

8
: An additional operand was specified

104
: Internal processing error

FORWARD Subcommand

```
►► Forward ►◄
```

Purpose

The FORWARD subcommand scrolls forward toward the highest address in the dump.

Usage Notes

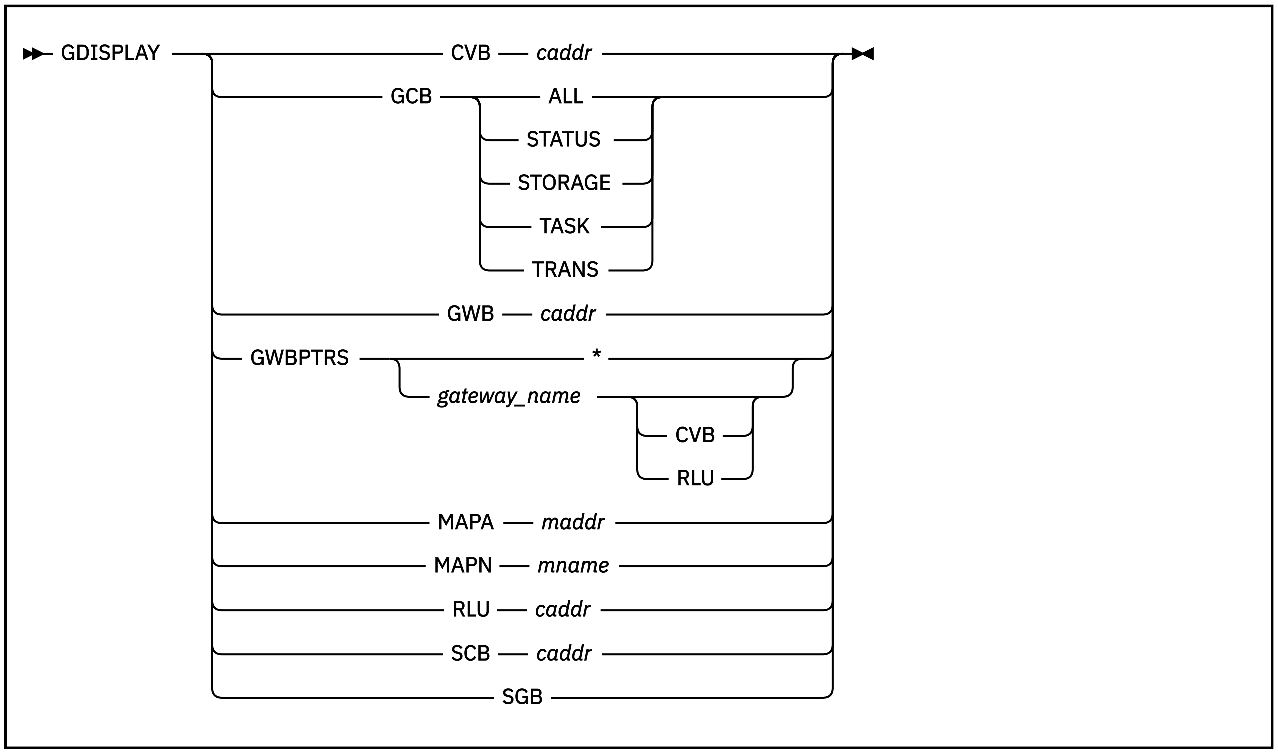
1. The FORWARD subcommand can be used after entering the DISPLAY, LOCATE, FINDMOD, TRACE, or other scrolling subcommands.
2. The FORWARD subcommand may be reentered by pressing ENTER (*null line* subcommand).
3. If the OFFSET operand was specified on the previous DISPLAY subcommand, issuing the FORWARD subcommand continues to display offsets.
4. You cannot display with offsets beyond X'FFF0'.
5. The FORWARD subcommand does not wrap the screen.
6. Refer to the BACKWARD subcommand to scroll backward toward the lowest address in the dump.

Responses

One full screen of the dump data is presented in both hexadecimal and EBCDIC.

When scrolling after the TRACE subcommand, the format of the next screen is identical with the format of the screen when TRACE was entered. For example, if the previous TRACE subcommand was for FORMAT output, scrolling continues with formatted output.

GDISPLAY Subcommand (AVS Dump)



Purpose

Use the GDISPLAY subcommand to display control blocks important to the AVS virtual machine and to display the module name and module address information for APPC/VM VTAM Support.

Operands

CVB *caddr*
displays the conversation block at control block address *caddr*.

GCB
displays the global control block (GCB).

ALL
displays all the information associated with the GCB.

STATUS
displays GCB ABEND, trace, and miscellaneous status information.

STORAGE
displays the GCB queue, stack, and ordered list parameters.

TASK
displays GCB task information.

TRANS
displays GCB transformation information.

GWB *caddr*
displays the gateway block at control block address *caddr*.

GWBPTRS
displays the addresses of the conversation block (CVB) or the remote LU block (RLU).

*****
displays the list of addresses for all gateway names.

gateway_name
is the name of an AVS gateway.

CVB
displays all the conversation block addresses for the specified *gateway_name*.

RLU
displays all the remote LU block addresses for the specified *gateway_name*.

MAPA *maddr*
displays the name of the AVS module located at address *maddr*.

MAPN *mname*
displays the address of *mname*, the 8-character name of an AVS module.

RLU *caddr*
displays the remote LU block at control block address *caddr*.

SCB *caddr*
displays the subtask control block at control block address *caddr*.

SGB
displays the scheduling global block.

Examples

Figure 30 on page 102 is an example of the output of the GDISPLAY subcommand. The following command produced this output:

```
gdisplay sgb
```

```
SCHEDULING GLOBAL BLOCK (SGB)

   CURRENT SUBTASK SCB ADDRESS  = 0000B8F8
   CURRENT SUBTASK PRIORITY     = 2
   VTAM SCB ADDRESS             = 0000B8D0
   IUCV SCB ADDRESS             = 0000B8F8
   COMMAND SCB ADDRESS          = 0000B880
   APPC/VM SCB ADDRESS          = 0000B8A8
   ACCOUNTING SCB ADDRESS       = 00000000
   MAXIMUM PRIORITY             = 2
   DISPATCH COUNTER             = 6
   ORIGINATORS SAVEAREA ADDRESS         = 00031FB0
   PRIORITY QUEUE TIME ARRAY ADDRESS    = 00037FF0
   PRIORITY QUEUE MARKER ARRAY ADDRESS  = 00037FF8
   PRIORITY QUEUE POINTER ARRAY ADDRESS = 00037FE8
```

Figure 30. Sample Output of the GDISPLAY Subcommand (SGB Operand)

Messages and Return Codes

AGWZTD440E
Operand missing or invalid

AGWZTR440E
Operand missing or invalid

AGWZTD441E
Conflicting operand: *operand*

AGWZMA442E
Address entered is not in AVS

AGWZMN443E
Name entered was not found in AVS

AGWZMA444E
AGWZAM table was not found

AGWZMN444E
AGWZAM table was not found

AGWZRS445E
The pointer to the *xxx* is invalid

AGWZSG445E
The pointer to the *xxx* is invalid

AGWZGW446E
The pointer to the *xxx* is not found

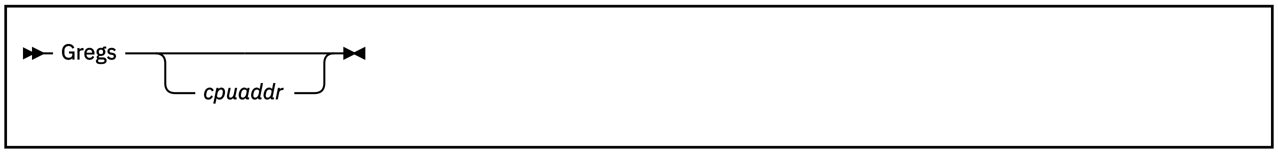
AGWZSG446E
The pointer to the *xxx* is not found

AGWZGW463E
Unable to list the control block addresses

AGWZGW464E
The pointer to the *named* ordered list is invalid

AGWZGW465E
The pointer to the *named* ordered list was not found

GREGS Subcommand



Purpose

Use the GREGS subcommand to display general purpose registers for a specified processor.

Operands

cpuaddr
is a 1- to 4-digit hexadecimal number specifying the physical address of the CPU whose general registers are to be displayed.

Usage Notes

1. If the *cpuaddr* operand is not specified, it defaults to the processor on which the CP VMDUMP command was issued for the virtual machine.
2. Use the CPU subcommand to obtain the CPU addresses in the dump.

Examples

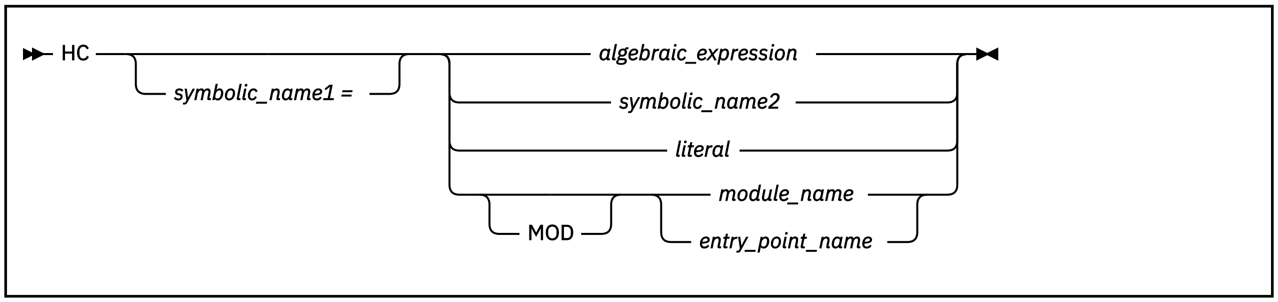
Figure 31 on page 104 illustrates the output of the GREGS subcommand for CPU address 0000. The subcommand entered is:

```
gregs
```

```
CPU ADDRESS - 0000
GENERAL REGS  0 - 15
     F0F0F0F0 F0F0F0F1 01488008 81CF6A44  00A4B3D8 00000002 01F3E708 4C4C4C4C
     00000001 00A4B3D8 00000002 00800000  01CF69A0 01F3C980 81CF6A82 0098A968
```

Figure 31. Sample Output of a GREGS Subcommand

HC Subcommand



Purpose

Use the HC subcommand (macro) to resolve hexadecimal calculations and algebraic expressions. HC may be entered as a subcommand from the DUMPSCAN command line or called as a REXX function from another macro.

Operands

***symbolic_name1* =**
: defines a symbolic variable for the DUMPSCAN session. The value for the symbolic name is assigned from the right side of the equal sign. HC symbolic variables follow the same naming conventions as those for REXX variables. Symbolic names are valid only for the HC subcommand.

algebraic_expression
: is an expression containing more than one of the following:

- *literal*
- *symbolic_name2*
- *module_name* or *entry point name*.

symbolic_name2
: is a name defined from a previous invocation of the HC subcommand.

literal
: is a 1- to 8-digit hexadecimal number.

MOD
: is an optional keyword indicating the next operand is a module name or entry point name. Use MOD to override the default precedence order defined in Usage Note 1.

module_name
: is an alphanumeric string of 1-to-8 characters that specifies a module name.

entry_point_name
: is an alphanumeric string of 1-to-8 characters that specifies an entry point name.

Usage Notes

1. The HC macro resolves hexadecimal literals, symbolic names, module names, entry point names and user IDs as terms. The following method is used:
   a. If the term was previously defined as a symbol on the HC subcommand, its value is substituted into the term.
   b. If the term is not a defined symbolic name, and the data type of the string is hexadecimal, the term will be treated as a hexadecimal literal.
   c. If the term is not a hexadecimal number, an attempt will be made to resolve the string as a module name or entry point name. This substitution is attempted only if the dump has a module

map appended. If the module name or entry point name is found, its address within the dump is substituted into the term.

d. If the term is not a predefined symbol, hexadecimal literal, module name, entry point name, or user ID, HC considers the string to be invalid. An error message is issued and the calculation is terminated.

Figure 32 on page 106 summarizes the description above.

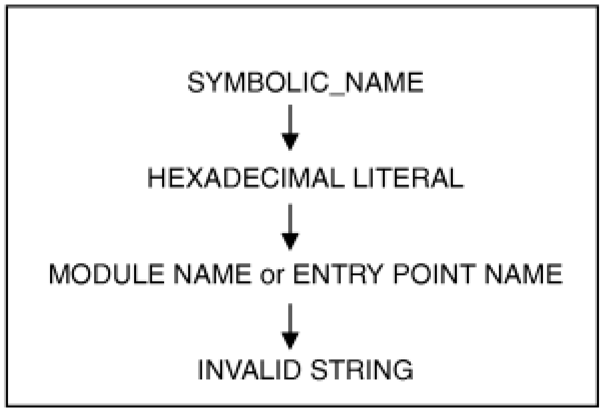


Figure 32. Order of String Resolution for the HC Subcommand (Macro)

2. HC supports the following operators:

   + (hexadecimal add)
   - (hexadecimal subtract)
   * (hexadecimal multiply)
   / (integer divide)

   The expression delimiters (and) are also supported.

3. To release a defined symbol name of its value, assign the symbol name to itself with HC. This is essentially a reset function. The following HC invocations illustrate a module name set to a symbolic name and then reset to its original value.

```
----> hc dmssop
009ED480
----> hc dmssop = 1+2
DMSSOP=3
----> hc dmssop
3
----> hc dmssop = dmssop
DMSSOC=DMSSOP
----> hc dmssop
009ED480
```

4. Imbedded blanks may surround didactic operators, equal sign ('='), and parenthesis (open and closed).

5. The HC and &*name* subcommands are independent of each other. The names defined in the &*name* subcommand may not be used in HC.

Examples

Figure 33 on page 107 illustrates the output you receive when you enter the HC subcommand (macro) for a symbolic name that is assigned an algebraic expression consisting of a module name plus a hex literal.

The subcommand entered for Figure 33 on page 107 is:

```
hc callsop = dmssfp+428
```

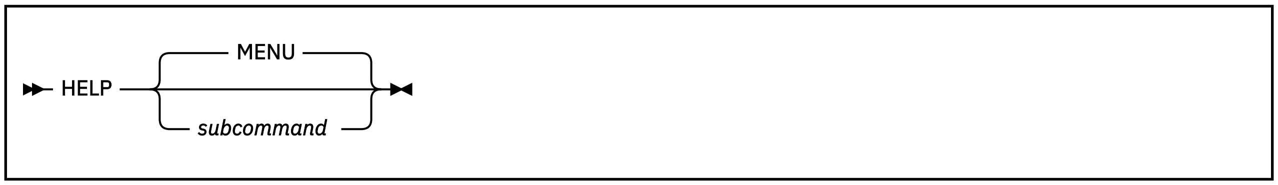
```
CALLSOP=0f414284
```

Figure 33. Sample Output of an HC Subcommand (Macro)

Messages and Return Codes

Return Code	Explanation
0	Successful execution
4	Dump has no module map
8	Syntax error on issuing HC
12	Invalid term
32	Internal error

HELP Subcommand



Purpose

Use the HELP subcommand to display a summary of the DUMPSCAN subcommands. The HELP subcommand describes the specified subcommand, including syntax, operands, and any relevant usage notes. If you do not enter a subcommand name, a list of all DUMPSCAN subcommands is presented.

Operands

MENU
displays a list of all DUMPSCAN subcommands. This is the default.

subcommand
can be any DUMPSCAN subcommand name and causes a description of the subcommand to be displayed. If the subcommand is not a DUMPSCAN subcommand, a list of all DUMPSCAN subcommands are presented.

Usage Notes

1. If you do not enter a subcommand name, or if you enter an invalid subcommand, a list of subcommands is displayed.
2. DUMPSCAN subcommand abbreviations can be used in the *subcommand* operand.

Examples

The HELP subcommand provides online information about command syntax, formats, and usage notes. The HELP text displayed is the same as displayed when HELP is invoked from CMS.

To display a menu of DUMPSCAN subcommands, enter:

```
help
```

To display the HELP text of a specific subcommand (for example, for the REGS subcommand) enter:

```
help dump regs
```

Messages and Return Codes

Return Code
Explanation

0
Normal

6
A subcommand has been rejected in the profile because of a LOAD error, or a QUIT subcommand has been issued in a macro called from the last file in the ring

11 & above
Standard CMS HELP command return codes

Error messages are issued from the CMS HELP Facility.

HX Subcommand

```
►►─ HX ─►◄
```

Purpose

Use the HX subcommand to end the session and return to CMS.

Usage Notes

1. The HX subcommand is equivalent to the QUIT subcommand.

Responses

CMS ready message.

IUCV Subcommand (GCS,AVS,RSCS Dumps)

```
►► IUcv ►◄
```

Purpose

Use the IUCV subcommand to display all entries in the IUCV path table. The IUCV path table contains information about all the IUCV paths in this virtual machine.

Examples

```
USER-ID-BLOCK  EXIT-ADDR  USER-WORD  TASK-BLOCK  PATH-STATUS

HHHHHHHH       HHHHHHHH   HHHHHHHH   HHHHHHHH    HHHH
```

Figure 34. Sample Output of the IUCV Subcommand

Responses

Displays for each path:

- Owner's ID block address
- Exit address
- User word
- Task control block address
- Path status.

Messages and Return Codes

GCTIIU31S
Insufficient free storage is available

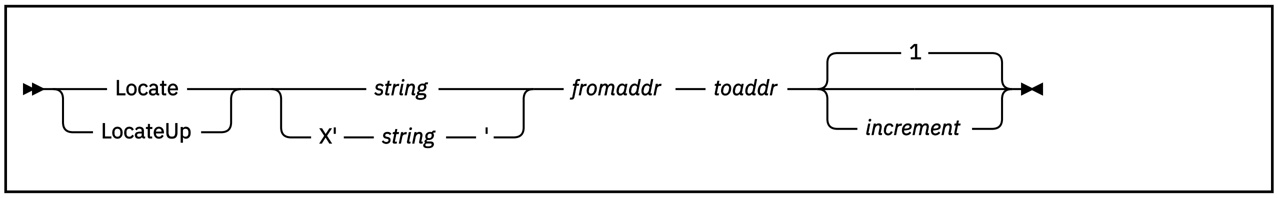
GCTIIU503I
No IUCV PATH table

GCTIIU504I
Page '*nnnnnnnn*' not found in dump

GCTIIU542I
IUCV anchor block ptr is zero. Cannot find IUCV path table

GCTIIU544I
IUCV PATH table ptr is zero

LOCATE(UP) Subcommand



Purpose

Use the LOCATE subcommand to search the dump for a particular string of data.

Operands

string
: is a string of up to 8 EBCDIC characters to be searched for.

X'*string*'
: is a 1- to 16-digit hexadecimal string to be searched for. The string must be in quotation marks and preceded by the letter X.

fromaddr
: is the 31-bit (4-byte) hexadecimal starting address for the search.

toaddr
: is the 31-bit (4-byte) hexadecimal number that is the ending address for the search.

increment
: is a 1- to 4-digit hexadecimal number to change the current address after each match attempt. 1 is the default increment if none is specified.

Usage Notes

1. All EBCDIC strings are truncated on the right to 8 characters. All hexadecimal strings are truncated on the right to 16 hexadecimal digits.
2. The second quotation mark of the X'*string*' operand is optional.
3. The LOCATE subcommand may be reissued by pressing ENTER (*null line* subcommand). The value of *fromaddr* is updated using the current address and the increment, and the subcommand is then reissued.
4. If the following conditions are true, using the *increment* operand in the LOCATE subcommand can reduce search time by eliminating unwanted matches.

   - If the target string is at a fixed displacement in each entry of a data area, and each entry has a fixed length
   - If the target string is at a fixed boundary (for example, fullword, doubleword, 16-byte, or 32-byte).

   For example, to check the beginning of each hexadecimal 20-byte entry from address X'4000' to X'8000' for the character string ABCD, enter:

   ```
   locate abcd 4000 8000 20
   ```

   The data at the hexadecimal addresses 4000, 4020, 4040, ... 8000 is searched for the string ABCD until the first occurrence (if any) is reached. These addresses are the increment length (X'20') apart.
5. The start of the string must be within the address range specified by the *fromaddr* and *toaddr* addresses. If the *fromaddr* and *toaddr* addresses are not specified, they will default to the beginning and ending of the dump.
6. The valid increment range is from X'1' to X'1000'.

7. If the LOCATE subcommand is placed in the &name table, the maximum string of 8 characters includes the hexadecimal identifier X with the hexadecimal characters within quotation marks (for example, X'13AB4').
8. If LOCATE is specified, the starting address must be less than the ending address; otherwise, an error message is issued.
9. If LOCATEUP is specified, the starting address must be greater than the ending address. If it is not, an error message is issued.

Examples

Figure 35 on page 112 illustrates the screen displayed when the subcommand LOCATE X'da441ea3' 50000 60000 8 is entered. This subcommand searches the dump from hex address 50000 up through address 60000 for the hexadecimal string X'da441ea3'. The dump is stepped through by adding 8 bytes from the current address until either a match is found or the *toaddr* is reached.

Note that the first occurrence of the string is on the first line of data at address 0005AD30.

The subcommand entered is:

```
lo x'da441ea3' 50000 60000 8
```

```
DISPLAY 0005AD30
  0005AD30  DA441EA3  50A0D670  41A09048  50A0D664 06 *...t&.O.....&.O.*
  0005AD40  58709044  06705070  D5ECD201  D6DA9040    *......&.N.K.O.. *
  0005AD50  D200D683  904217AA  BFA19043  42A0D69C    *K.Oc.........O.*
  0005AD60  4CA0D678  5830B1A0  1EA3D203  D5B0A008    *<.O......tK.N...*
  0005AD70  BF8FD5B0  47F0C3B6  BF8FD5B0  4780C27A    *..N..0C...N...B:*
  0005AD80  58F0DA40  0DEF41E0  C2D850F0  E0544770    *.0. ....BQ0...*
  0005AD90  C27A9508  80004780  C3589520  80004770    *B:n.....C.n.....*
  0005ADA0  C27A9120  80014710  CDCC9210  DA0B5870    *B:j.......k.....*
  0005ADB0  B1D8BF1F  70004780  C2C84177  00085880    *.Q......BH......*
  0005ADC0  70205980  D5B04780  C2C058F0  DA400DEF    *....N...B..0. ..*
  0005ADD0  41000000  4780C2AC  41000001  12FF4780    *......B.........*
  0005ADE0  C2BA41E0  C2D850F0  E0541200  4780CDCC    *B...BQ0.......*
  0005ADF0  41707090  4610C28E  4110D7A0  4590D326    *......B...P...L.*
  0005AE00  47F0D136  00000000  00000000  00000000    *.0J.............*
  0005AE10  00000000  00000000  00000000  00000000    *................*
  0005AE20  A1C18752  00021900  00001000  0001F900    *.Ag...........9.*
  0005AE30  0001E900  00001FE0  00000000  00000000    *..Z.............*
  0005AE40  00014000  00014000  00001000  00000000    *.. ... .........*
  0005AE50  0011B000  00000008  8011B3FE  00000000    *................*
  0005AE60  00000000  00000000  00000000  00000000    *................*
  0005AE70  00000000  00000000  00000000  00000000    *................*
  0005AE80  00000000  00000000  95108001  4780C378    *........n.....C.*
  0005AE90  95808001  4780C378  95208001  4780C378    *n.....C.n.....C.*
  0005AEA0  95408001  4770C27A  D204D865  DAAB45E0    *n ....B:K.Q.....*
  0005AEB0  D35E4100  D4E89540  80014780  C3924100    *L;..MYn ....Ck..*
  0005AEC0  D4D858F0  DA480DEF  4770D136  4110DA8A    *MQ.0......J.....*
  0005AED0  58F0DA4C  0DEF4770  D1365890  D5BC9202    *.0.<....J...N.k.*
  0005AEE0  939A47F0  C46645E0  C73A4740  C27A4710    *l..0D...G.. B:..*
  0005AEF0  C2C8D204  D865DAB0  45E0D35E  5890D5BC    *BHK.Q.....L;..N.*
  0005AF00  9203939A  5810D5B4  D2051048  D848D201    *k.l...N.K...Q.K.*
  0005AF10  D6DA102C  5830D608  4160D598  92066000    *O.....O..-Nqk.-.*
  0005AF20  45E0CB22  4160D598  92056000  41103100    *.....-Nqk.-.....*
  0005AF30  5010D5F8  D203D624  D680D201  D628D6DA    *&.N8K.O.O.K.O.O.*
  0005AF40  D203D630  D61CD200  D634DA0A  D201D636    *K.O.O.K.O...K.O.*
  0005AF50  D6ECD201  D638D6EE  5080D62C  D203D63C    *O.K.O.O.&.O.K.O.*
  0005AF60  D680D201  D640D6DA  D203D648  D61CD200    *O.K.O O.K.O.O.K.*
```

Figure 35. Sample Output of a LOCATE Subcommand with the Increment Operand

Responses

Provided the end of dump has not been reached, one full screen of data is presented, in both hexadecimal and EBCDIC. The target string is positioned on the first line at the hexadecimal location where the string begins.

OSPOINT Subcommand (CMS Dump)

```
►► OSPoint ►◄
```

Purpose

Use the OSPOINT subcommand to display the formatted contents of three pointers used in OS simulation.

Examples

The following figure is an example of the output of the OSPOINT subcommand.

```
CVTSECT= 00002C58
FCBSECT= 00000000
OPSECT = 00003BA0
```

Figure 36. Example Output of the OSPOINT Subcommand

Responses

OSPOINT displays the formatted contents of the following OS simulation pointers:

CVTSECT
Address of the simulated communications vector table

FCBSECT
Address of the first file control block

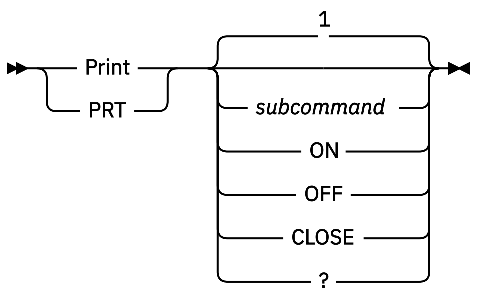
OPSECT
Address of the reading and writing parameter list

Messages and Return Codes

DMSDFR2017I
INVALID OPERAND - *operand*

DMSDOP2017I
PAGE '*page*' NOT FOUND IN DUMP

PRINT Subcommand



Notes:

[1] The default is the last subcommand that was entered.

Purpose

Use the PRINT subcommand to print data displayed on your terminal by one of the DUMPSCAN subcommands.

Operands

subcommand
is a DUMPSCAN subcommand to be entered. Its results are printed and displayed.

ON
turns on the print switch to collect data for printing.

OFF
turns off the print switch, but does not close the virtual printer.

CLOSE
closes the current print file in the virtual printer, but does not turn the print switch off.

?
displays the print switch status (ON or OFF).

Usage Notes

1. When the print switch is ON, all data displayed at the terminal is also written to the virtual printer.
2. PRINT with no operands reissues the subcommand previously entered and prints the data. The data is not redisplayed at the terminal.
3. Synonyms for the PRINT subcommand are not allowed in the &name table. (See the &name subcommand in Chapter 5, “DUMPSCAN Subcommand Reference,” on page 51.)
4. CLOSE is automatically issued at the end of the DUMPSCAN session.
5. If the print switch is OFF, the PRINT subcommand turns it on for the subcommand’s operation, then turns it off. If the print switch is ON, it is left on.
6. PRT is a synonym for the PRINT subcommand.
7. Each line of output in the print file has a prefix field showing which dump file produced the data.

QUIT Subcommand

```
►►─ QUIT ─►◄
```

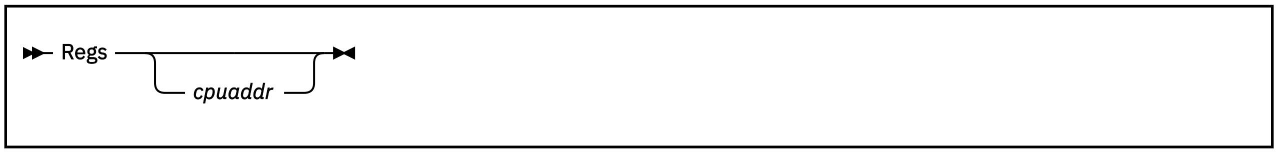
Purpose

Use the QUIT subcommand to end the session and return to CMS.

Usage Notes

1. The QUIT subcommand is equivalent to the HX subcommand.

REGS Subcommand



Purpose

Use the REGS subcommand to display registers, clocks, timer, and program status words for a specific processor.

Operands

cpuaddr
is a 1- to 4-digit hexadecimal number specifying the CPU address for which the information is to be displayed.

Usage Notes

1. For 370-mode virtual machine dumps, the output of the REGS subcommand also includes:
   - Channel status word (CSW)
   - Channel address word (CAW)
   - Interval timer
   - Current program status word (PSW). For CP abend dumps, the current PSW is available in the store status area of the prefix page. For CP stand-alone dumps, the store status information for the IPL CPU is located in absolute page 0.
2. The REGS subcommand clears the screen prior to presenting data.
3. Use the CPU subcommand to obtain the CPU addresses in the dump.
4. If the *cpuaddr* operand is not specified, it defaults to the CPU on which the CP VMDUMP command was entered for the virtual machine.
5. The REGs subcommand does not display access registers for 370-mode virtual machine dumps.
6. For virtual machine dumps, the REGS subcommand formats the crypto domain index register for ESA virtual machines that have the Integrated Cryptographic Facility defined.

Examples

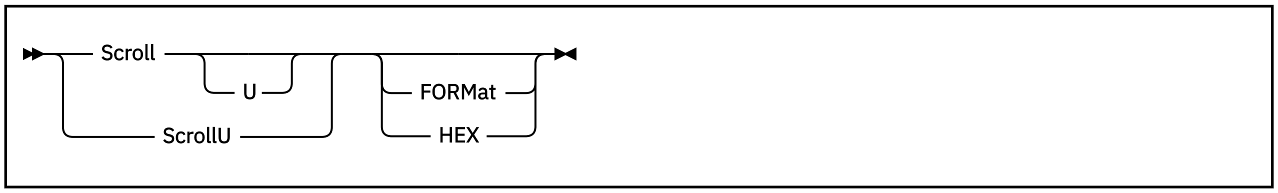
Figure 37 on page 117 illustrates the output of the REGS subcommand for a virtual machine dump. The subcommand entered is:

```
regs
```

```
REGS
CPU ADDRESS - 0000                        PREFIX REGISTER - 00000000
GENERAL REGS  0 - 15
     D43F372C 00005964 00000017 000057E0  000017F8 00000000 00002025 000017F8
     000057E0 00E01352 00000006 0B3B6582  00EC5B98 003F36E0 50EC5C36 00E01352
CONTROL REGS  0 - 15
     010000E0 00000000 FFFFFFFF 00000000  00000000 00000000 00000000 00000000
     00000000 00000000 00000000 00000000  00000000 00000000 C2000000 00000200
ACCESS  REGS  0 - 15
     00000000 01000003 01000003 00000000  00000000 01000003 00000000 00000000
     00000000 01000003 00000000 00000000  00000000 00000000 00000000 00000000
FLOATING POINT REGS  0 - 6
     00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000

TOD CLOCK         A211E668 FDF4CA01                CSW 00004AB8 0C000000
CLOCK COMPARATOR  A211D917 FF495000                CAW 00004AA8
CPU TIMER         FFFFE3CB 60D8CB00
INTERVAL TIMER    00000000

EXT OLD FF044001 00E5EDEA  INT CODE 4001            EXT NEW 00000000 00E2D930
SVC OLD FFE000CA 5003B30C  INT CODE 00CA ILC 0002   SVC NEW 00000000 0044D630
PGM OLD 00000004 D0E7328A  INT CODE 0004 ILC 0006   PGM NEW 00000000 00E30238
MCH OLD 00000000 00000000                           MCH NEW 00000000 00E412F8
I/O OLD FFFFFFFF FFFFFFFF                           I/O NEW 00000000 00E05360
```

Figure 37. Sample Output of a REGS Subcommand for a Virtual Machine Dump

SCROLL Subcommand



Purpose

Use the SCROLL subcommand to repeat the most recent TRACE subcommand with an adjusted address for server virtual machine dumps.

SCROLLU decreases the target address to display the preceding screen of data. SCROLL increases the target address to display the next screen of data.

Operands

FORMat
displays the trace entries in the long (FORMAT) version. This operand is valid only when you enter the SCROLL subcommand after a formatted display of the trace entries.

HEX
displays the trace entries in the short (HEX) version. This operand is valid only when you enter the SCROLL subcommand after a formatted display of the trace entries.

Usage Notes

1. The SCROLL subcommand is intended for use on a display terminal and scrolls from the last hex entry displayed.
2. The SCROLLU subcommand scrolls from the first entry displayed; it displays the preceding addresses.
3. You can reissue the SCROLL and SCROLLU subcommands by pressing ENTER or its equivalent.
4. The HEX and FORMAT options are invalid for SFS dumps.

Examples

The following is a valid sequence of TRACE subcommands:

To display a screen of trace entries, setting the FROM address and the FORMAT option, you enter the command:

```
trace format
```

To display the next screen of formatted (FORMAT) trace entries, you enter the command:

```
scroll
```

To change the HEX or FORMAT setting while continuing to scroll, you can enter the SCROLL subcommand and specify HEX or FORMAT operands. For example, you enter either of the following commands:

```
scroll u format
```

or

```
trace scroll u format
```

The SCROLL option of the TRACE subcommand returns control to the trace formatter after using the DISPLAY subcommand to display storage. For example, you might do the following:

- Display trace entries
- Use the SCROLL subcommand to format and display more trace entries
- Enter a DISPLAY subcommand to look at storage
- Enter a SCROLL subcommand.

However, the storage will not be formatted as trace entries, even if the displayed area is in the trace table. Because the SCROLL subcommand is not processing trace entries at this time, the FORMAT and HEX operands are not valid operands. The following is an example of an invalid sequence:

To display a screen of trace entries, setting the FROM address and the FORMAT, enter the command:

```
trace format
```

To display a screen of trace entries with the HEX operand, you enter:

```
scroll u hex
```

To look at this area of storage in display format, enter:

```
display 240000
```

To look at the next screen of displayed storage, enter the command:

```
scroll
```

When you enter the next command, you receive error message DMMSCR863E, which states you have entered an invalid parameter:

```
scroll format
```

The SCROLL option on the TRACE subcommand, explained in "TRACE Subcommand" on page 129, lets you return to formatting trace entries after a display of some storage area. The following example contains a valid sequence of commands.

To display a screen of trace entries, setting the FROM address and the FORMAT, you enter the following command:

```
trace format
```

To display a screen of trace entries with the HEX operand, enter:

```
scroll u hex
```

To look at an area of storage in display format, you enter:

```
display 240000
```

To look at next screen of displayed storage, you enter:

```
scroll
```

Finally, to look beyond last-displayed trace entries and use the FORMAT option, you enter the command:

```
trace scroll u format
```

Responses

SCROLL displays a full screen of storage data (hexadecimal and translated) or trace data in the existing format mode. The current line pointer (line 10) is positioned X'130' bytes from its previous location.

Messages and Return Codes

ATSZTO076E
FORMATTED DATA ENTRY EXCEEDS MAXIMUM SIZE

ATSZTS087E
ATTEMPT TO GO BEYOND STORAGE BOUNDARY

SYMPTOM Subcommand

```
►► SYMPtom ►◄
```

Purpose

Use the SYMPTOM subcommand to display formatted symptom record information at your terminal.

Usage Notes

1. If the dump symptom record is missing or not readable, you receive an error message.
2. For virtual machine dumps only, an additional line of output is displayed for the Section 5 Data: the number of address spaces, including the primary, that were dumped by the CP VMDUMP command or by DIAGNOSE code X'94'.

   See Figure 38 on page 121 for an example.

   This information is useful for locating other related dump files of address spaces. The dump files created by DUMPLOAD have file types numbered in sequence, beginning with DUMP0001. The Dump Viewing Facility will not be able to locate as many files as SYMPTOM reports if DUMPLOAD failed while it was building the dump files, or if any of the related dump files have been renamed.

   See "Virtual Machine Dumps in an XC Environment" on page 11 for more information.
3. SYMPTOM output may span several screens. Use the XEDIT subcommands FORWARD and BACKWARD to scroll the display.

Examples

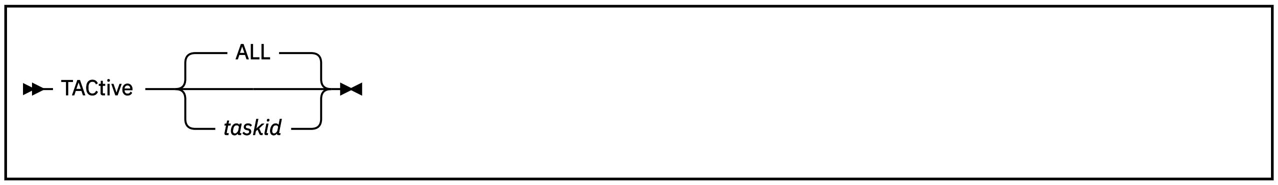
Figure 38 on page 121 shows an output screen for a VM dump when the SYMPTOM subcommand is entered:

```
symptom
```

```
           SYMPTOM RECORD FOR INCIDENT   A1A2351C CD3D0SYM

 TOD CLOCK . . A1A2351CCD3D0E01                DATE. . . . . 02/08/99
 TIME ZONE . . -05:00:00                       TIME. . . . . 17:59:25

 CPU MODEL . . 9672                            BASE SCP. . . 5654
 CPU SERIAL. . 174554                          NODEID. . . . VMESA

 DUMP  NAME: HUNGSYS2 DUMP0001                 DUMP TYPE . . VMDUMP
 ------------------------------------------------------------------------
 SECTION 5 DATA:
               USERID DUMPED: USER1
               DUMP RECEIVER: USER1
               SPOOLID: 0920
               NAME OF MAIN DUMP: HUNGSYS2 DUMP0001
               NUMBER OF ADDRESS SPACES DUMPED: 0006
```

Figure 38. Output Format of a SYMPTOM Subcommand for a VM Dump

TACTIVE Subcommand (GCS,AVS,RSCS Dumps)



Purpose

Use the TACTIVE subcommand to display the task's active program list.

Operands

taskid
identifies the task you want information about. The format is *nnnn*.

ALL
requests information for all tasks. ALL is the default.

Examples

Figure 39 on page 122 shows sample output for the TACTIVE subcommand.

```
TASK   BLOCK   COMPLETION   BLOCK   TYPE   PROGRAM   ENTRY
 ID   ADDRESS     CODE     ADDRESS           NAME   ADDRESS

HHHH  HHHHHHHH   HHHHHH   HHHHHHHH   HH     EEEEE    HHHHHHH

0001    002438   000000     16B600   40     GDUMP     1870E6
                     R0 =001690D4 R1 =001690F8 R2 =00169310
                     R3 =00000000 R4 =00000590 R5 =00000000
                     R6 =00000678 R7 =00000678 R8 =00000006
                     R9 =002FD000 R10=00169048 R11=00009F38
                     R12=50182F1C R13=00009F28 R14=50183280
                     R15=00000001
                            002518    80    CONSOLE   182EA8
                     R0 =00000000 R1 =00000000 R2 =00000000
                     R3 =00000000 R4 =00000000 R5 =00000000
                     R6 =00000000 R7 =00000000 R8 =00000000
                     R9 =00000000 R10=00000000 R11=00000000
                     R12=00000000 R13=00000000 R14=00000000
                     R15=00000000

  TASK   BLOCK  COMPLETION    BLOCK          PROGRAM   ENTRY
   ID   ADDRESS    CODE     ADDRESS  TYPE     NAME    ADDRESS
0002    002430   000000     0035B0    80     COMMAND   182A68
                     R0 =00000000 R1 =00000000 R2 =00000000
                     R3 =00000000 R4 =00000000 R5 =00000000
                     R6 =00000000 R7 =00000000 R8 =00000000
                     R9 =00000000 R10=00000000 R11=00000000
                     R12=00000000 R13=00000000 R14=00000000
                     R15=00000000
```

Figure 39. Sample Output of the TACTIVE Subcommand

Responses

Displays a chart containing the task ID, the address of the task control block, and the task completion code. A state block is a control block that contains information about an active program. There are three types of state blocks:

- Link blocks represent programs that have been invoked through the LINK, SYNCH, XCTL, or ATTACH macros, or the OSRUN command
- SVC blocks represent calls to the SVC interrupt handler

- Asynchronous exit blocks exist for asynchronous exits scheduled for this task.

For every state block on the task's active program list, this subcommand also displays the:

- Address of the state block
- Type of state block (link block, SVC block, or asynchronous exit block)
- Name and entry-point address of the program that the block represents
- Register contents associated with the state block.

Messages and Return Codes

GCTIAL031S
Insufficient free storage is available

GCTIAL504I
Page '*nnnnnnnn*' not found in dump

GCTIAL505I
TASKID '*xxxxxxxx*' invalid

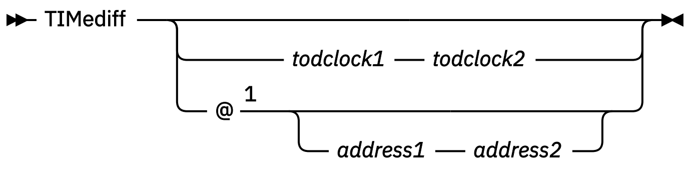
GCTIAL545I
NUCON extension pointer is zero. Cannot find state block

GCTIAL546I
Task block PRT is zero. Cannot find state block

GCTIAL547I
State block PRT is zero

GCTIAL548I
Taskid table PRT is zero. Can't find state block

TIMEDIFF Subcommand



Notes:

[1] Do not put blanks between the operands and special characters.

Purpose

Use the TIMEDIFF subcommand (macro) to display the difference in time between two TOD clock values.

Operands

todclock1
is a 1- to 16-digit hexadecimal number specifying the first TOD clock value.

todclock2
is a 1- to 16-digit hexadecimal number specifying the second TOD clock value.

@
indicates addresses are used specifying the TOD clock values. Eight bytes are used as the TOD clock value.

address1
is the 31-bit (4-byte) hexadecimal address from which the first TOD clock is read.

address2
is the 31-bit (4-byte) hexadecimal address from which the second TOD clock is read.

Usage Notes

1. Values are treated as the rightmost positions of a 16-digit TOD clock. So, for example, a TOD clock comparison value of 68A23451 would be equivalent to 0000000068A23451.
2. If the two comparison TOD clock values are of different length, leading digits of the longer operand are ignored.
3. You may specify both TOD clock values from cursor positions on the screen. To use screen TOD clock values, type the subcommand with no operands (TIMEDIFF), then position the cursor at the screen position where you want data substitution for the first TOD clock. Press Enter, and you will be prompted to position the cursor for screen data substitution for the second TOD clock. Position the cursor and press Enter. The subcommand is completed.
4. You may specify both TOD clock addresses from cursor positions on the screen. To use screen address values, type the subcommand with the address keyword and no *address* operands (TIMEDIFF @), then position the cursor at a screen position where you want data substitution for the first address. Press Enter, and you will be prompted to position the cursor for screen data substitution for the second address. Position the cursor and press Enter. The subcommand is completed.
5. The following rules apply for entering TOD clock values from screen positions:
   - TOD clock values are delimited on the left by a blank or column 1. If you place the cursor in the middle of a string, TIMEDIFF will shift it left until either a blank is found or column one reached.
   - TOD clock values are delimited on the right by a blank or the first 16 characters found.
   - It is invalid to have nonhexadecimal digits in the specified string.

6. When selecting addresses from screen positions with the @ operand:
   - Addresses are delimited on the left by a blank or column 1. If you place the cursor in the middle of a string, TIMEDIFF will shift it left until either a blank is found or column one reached.
   - It is invalid to have nonhexadecimal digits in the specified string.
   - Only the lower 3 bytes are used in the difference calculation when the TOD clocks values are read from the trace tables.

Examples

Figure 40 on page 125 illustrates the output of the TIMEDIFF subcommand (macro). The subcommand entered is:

```
timediff 69e93bd68000 69e93bb8e000
```

```
time1 = 69E93BD68000
time2 = 69E93BB8E000
time2 is 474.00000 microsecs before time1
```

Figure 40. Sample Output of a TIMEDIFF Subcommand (macro)

Figure 41 on page 125 illustrates the output of the TIMEDIFF subcommand (macro) entered with address operands. The subcommand entered is:

```
timediff @1A2000 1A2060
```

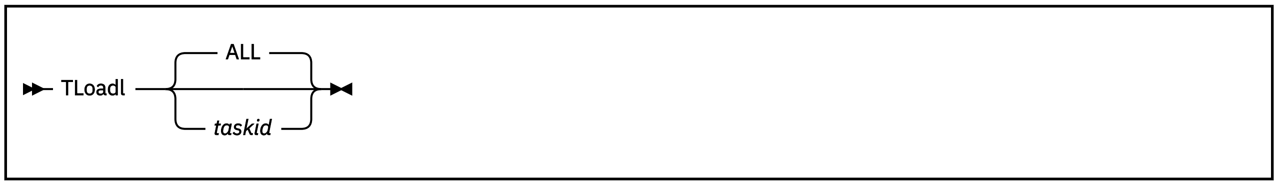
```
time1 = 1A2000 -> 69E93BD68000
time2 = 1A2060 -> 69E93BB8E000
time2 is 474.00000 microsecs before time1
```

Figure 41. Sample Output of a TIMEDIFF Subcommand (macro) entered with addresses

Messages and Return Codes

Return Code	Explanation
0	Successful completion
4	Invalid TOD clock value
8	Syntax error on issuing TIMEDIFF
12	Address specified is not in the dump
32	Internal error

TLOADL Subcommand (GCS,AVS,RSCS Dumps)



Purpose

Use the TLOADL subcommand to display the task load list.

Operands

taskid
identifies the task you want information about. The format is *nnnn*.

ALL
requests information for all tasks. ALL is the default.

Examples

Figure 42 on page 126 shows an example of the output of the TLOADL subcommand.

```
TASK-ID  TASK-BLOCK  LOAD-BLOCK  PROGRAM-NAME  LOAD-COUNT

HHHH     HHHHHHHH    HHHHHHHH    EEEEEEEE      HHHH
 .          .           .           .           .
 .          .           .           .           .
 .          .           .           .           .
```

Figure 42. Sample Output for the TLOADL Subcommand

Responses

Displays the following for each program loaded by this task:

- The address of the control block that points to where the program is loaded
- The associated program name
- The number of times it has been loaded, but not deleted.

Messages and Return Codes

GCTITL031S
Insufficient free storage is available

GCTITL504I
Page '*nnnnnnnn*' not found in dump

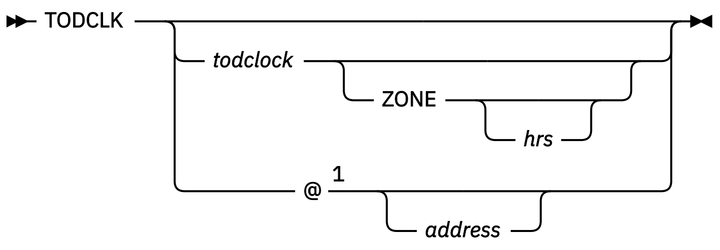
GCTIAL505I
TASKID '*xxxxxxxx*' invalid

GCTITL535I
TASKID table PRT is zero. Cannot find task load list

GCTITL537I
Task block PRT is zero. Cannot find task load list

GCTITL538I
Task block list PRT is zero

TODCLK Subcommand



Notes:

[1] Do not put blanks between the operands and special characters.

Purpose

Use the TODCLK subcommand (macro) to display the date and time for a specified hexadecimal TOD clock value.

Operands

todclock
is a 1- to 16-digit hexadecimal value.

ZONE
is a keyword used to obtain the local time.

hrs
is a 1- to 4-character decimal time zone difference to be applied to the TOD clock value, which is based on GMT. If *hrs* is not specified with the keyword ZONE, the differential is taken from the dump symptom record. The *hrs* operand may be specified as a signed decimal such as -5.5 or -5.

@
indicates an address is used specifying the TOD clock value. Eight bytes read from the dump at the address are used as the TOD clock value.

address
is the 31-bit (4-byte) hexadecimal address from which the TOD clock data is read.

Usage Notes

1. If the supplied TOD clock value is less than 16 digits, the value is padded to the right with zeros before being converted into date and time.
2. You may specify TOD clock values from cursor positions on the screen. To substitute a screen value as a TOD clock, type the subcommand with no operands (TODCLK), then position the cursor at a screen position and press Enter.
3. You may specify addresses from cursor positions on the screen. To substitute a screen value as an address, type the subcommand with the address keyword and no *address* operand (TODCLK @), then position the cursor at a screen position and press Enter.
4. The following rules apply for entering TOD clock values from screen positions:

   TOD clock values are delimited on the left by a blank or column one. If you place the cursor in the middle of a string, TODCLK will shift it left until either a blank is found or column one is reached.

   TOD clock values are delimited on the right by a blank or the first 16 digits found.

   It is invalid to have nonhexadecimal digits in the specified string.

If TODCLK is entered without any operands, the TOD value is taken from the cursor position in the dump view file area. If the cursor is not positioned in the file area, the command puts the current time and date into the dump view file.

Examples

The following figures illustrate the output of the TODCLK subcommand (macro). The subcommand entered is:

```
todclk
```

```
Todays Date 01/21/88 Current Time 11:19:54
```

Figure 43. Sample Output of a TODCLK Subcommand (macro)

The subcommand entered is:

```
todclk 9df469ec5e511000
```

```
----> todclk 9df469ec5e511000
9DF469EC5E51100 +0 => Date 01/21/88 Time 11:19:54
```

Figure 44. Sample Output of a TODCLK Subcommand (macro)

The subcommand entered is:

```
todclk 9df469ec5e511000 zone 2.5
```

```
----> todclk 9df469ec5e511000 zone 2.5
9DF469EC5E51100 +2.5 => Date 01/21/88 Time 11:22:24
```

Figure 45. Sample Output of a TODCLK Subcommand (macro)

The subcommand entered is:

```
todclk @1FAA90 zone -5
```

```
----> todclk @1FAA90 zone -5
1FAA90 -> 9DF469EC5E51100 -5 => Date 01/21/88 Time 05:52:24
```

Figure 46. Sample Output of a TODCLK Subcommand (macro)

Messages and Return Codes

Return Code
Explanation

0
Successful execution

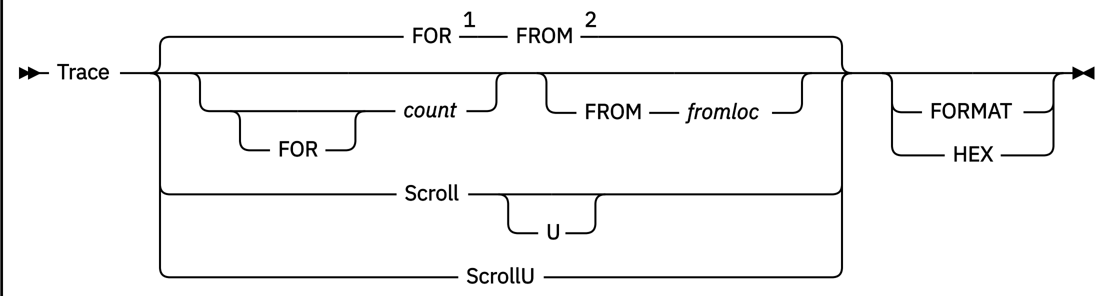
4
Invalid TOD clock value or hrs

8
Syntax error on issuing TODCLOCK

12
The specified address is not in the dump

32
Internal error

TRACE Subcommand



Notes:

[1] The default is a full screen of data on a 24-line terminal.

[2] The default value is the address of the most recent trace table entry.

Purpose

Use the TRACE subcommand to display trace table entries in either the short (HEX) or fully formatted versions.

After the first invocation of the TRACE subcommand, you may specify either the SCROLL option to move forward or backward through the trace table, using the formatting options that you established with an earlier TRACE subcommand. You can also enter the REUSE NULL LINE subcommand.

Operands

FOR
is an optional keyword operand. If you specify FOR, a *count* value must follow immediately. Do not combine this operand with the SCROLL or SCROLLU operand.

count
is a decimal number from 1 to 999 that specifies the number of entries to be displayed in the trace table. Do not combine this with the SCROLL or SCROLLU operand.

FROM
is an optional keyword operand. If you specify it, a *fromloc* value must follow immediately. Do not combine this with the SCROLL or SCROLLU operand.

fromloc
is a hexadecimal address from which the trace entries are displayed. You must specify a location that is on a 16-byte boundary.

Scroll
is an optional keyword operand. If you specify it, the next screen of trace entries is displayed. SCROLL U has the same effect as the SCROLLU operand. This option is only valid if you already entered a successful TRACE subcommand. Do not combine this option with the FROM or FOR options.

ScrollU
is an optional keyword operand. If you specify it, the preceding screen full of trace entries is displayed. This option is only valid if you already entered a successful TRACE subcommand.

FORMAT
is an optional keyword operand. If you specify it, formatted trace table entries are displayed. The FORMAT operand is invalid for SFS.

HEX
is an optional keyword operand. If you specify it, unformatted trace table entries are displayed in hexadecimal, along with a brief description of the entry. HEX is the default the first time you enter

TRACE in a DUMPSCAN session. Thereafter, it defaults to the previous formatting keyword (HEX or FORMAT). Do not combine this with the FORMAT operand. The HEX operand is invalid for SFS.

Usage Notes

1. The AVS, TSAF, and SFS trace formatting routines must be on an accessed disk.
2. The formatted output produced by the TRACE subcommand is valid for z/VM dumps only. Unpredictable results can occur in the formatted output if you use this function against a pre-z/VM dump. When analyzing data, you should rely only on the hexadecimal TRACE data and not the formatted or brief descriptions provided with the hexadecimal data.
3. If the trace table pointers are invalid, you can use the FROM operand and location values to view the trace table.
4. HEX and FORMAT operands may not be specified together. However, you may specify any of the operands together in any order.

Error Detection

The TRACE subcommand detects three types of errors:

- Invocation errors, which may consist of:
  - Conflicting operands
  - Missing operands
  - Nonhexadecimal characters in a hexadecimal operand
  - A *fromloc* address that is outside the range of addresses indicated in the valid trace table pointers
  - For AVS, SFS, and TSAF, a *fromloc* address that is not at the beginning of a valid trace entry
  - An invalid count operand (not within the range of 1 to 999).

- Errors that are detected in the dump, which include:
  - Invalid trace table pointers:
    - Start pointer is greater than the trace table stop pointer
    - Current pointer points are outside of the trace table
    - Start or stop pointer is not on a page boundary
    - Pointers do not point to valid trace entries
    - The map is not appended (TSAF only)
    - The trace table must be full pages.
  - A page that is to be used for the trace function is not in the dump. This page may contain the pointers or the trace entries.
- Necessary resources that are not available.

Examples

To display the last four entries in the trace table, where address X'CCB230' is the most recent entry, you enter either of the following commands:

```
trace 4 hex
```

or

```
trace for 4 from ccb230
```

The following output is produced; this hexadecimal display consists of:

- Trace entry address (first column)
- Hexadecimal display of the trace entry (second through fifth columns)
- Trace entry description (last column).

```
00CCB200    08FEBCC8 800C4202 03800000 0000F438    ENTER SCHEDULER
00CCB210    09FEBCC8 0000F1AB 084000BC 00F908FC    QUEUE DROP
00CCB220    03000000 00040040 000C2000 0002B8D2    PROGRAM INTERRUPT
00CCB230    02016F04 00020008 000C0000 00018D66    SVC INTERRUPT (CALL)
```

Figure 47. Example Output of the TRACE Subcommand (HEX Display)

Example (HEX TSAF and AVS)

Because TSAF and AVS have the same general format, only TSAF output is shown in this example.

To display the last seven entries in the TSAF trace table, where address X'D8EF' is the most recent entry, you enter either of the following commands:

```
trace 8 hex
```

or

```
trace for 8 from d8ef hex
```

The following output is produced; this hexadecimal display consists of:

- Trace entry address (first column)
- Hexadecimal display of the trace entry trailer record (middle column)
- Trace entry description (last column).

```
0000D82D    9AF3D043 1B168000 D3F1E3 A572 0008     ATSL1T EXIT
0000D846    9AF3D043 1B1D9000 E5E2D2 C00C 0029     DISPATCH A TASK
0000D880    9AF3D043 1B1F2000 D3D4D5 A168 0005     WOKE UP IN STATE
0000D896    9AF3D043 1B21D000 D4E3E8 60E1 0004     MODULE ENTERED AT ATSMTY
0000D8AB    9AF3D043 1B225000 D4E3E8 60E8 0005     TRACE INPUT
0000D8C1    9AF3D043 1B234000 D4E3E8 60E6 0008     ATSMTY EXIT
0000D8DA    9AF3D043 1B298000 D3F1D7 A461 0004     MODULE ENTERED AT ATSL1P
0000D8EF    9AF3D043 1B29D000 D3F1D7 A464 0005     LINK PURGED
```

Figure 48. Sample Output of the TRACE Subcommand (HEX TSAF Display)

Example (FORMAT TSAF and AVS)

Because TSAF and AVS have the same general format, only TSAF output is shown in this example.

To display three entries in the trace table, beginning with the most recent trace table entry (X'D8EF'), you enter either of the following commands:

```
trace 3 format
```

or

```
trace for 3 from d8ef format
```

```
60E6 ATSMTY EXIT                                               ADDR = 0000D8C1
       CLOCK = 9AF3D043 1B234000               MODULE = ATSMTY
       TIME = 08:43:08.627508 GMT 05/20/1986   LENGTH = 0008
       R14                6002F8E4                             *-.8U*
       R15                0004AB30                             *....*

A461 MODULE ENTERED AT ATSL1P                                  ADDR = 0000D8DA
       CLOCK = 9AF3D043 1B298000               MODULE = ATSL1P
       TIME = 08:43:08.627608 GMT 05/20/1986   LENGTH = 0004
       PARM_LIST          000300A4                             *...u*

A464 LINK PURGED                                               ADDR = 0000D8EF
       CLOCK = 9AF3D043 1B29D000               MODULE = ATSL1P
       TIME = 08:43:08.627613 GMT 05/20/1986   LENGTH = 0005
       LINK_NUMBER        00000004                             *....*
```

Figure 49. Sample Output of the TRACE Subcommand (FORMAT TSAF)

The formatted display consists of the following information:

- Main line containing trace:
  - Type number
  - Entry description
  - Table address.
- Two lines containing:
  - Time stamp for trace entry
  - Module that produced the trace entry
  - Time stamp in readable format
  - Length of the data files (in hex).
- Zero or more additional lines containing formatted information (with descriptors) from the trace entry.

Example (SFS)

To display the most recent trace table entry, you enter the command:

```
trace 1
```

If address X'47A8F7' is the most recent trace entry, TRACE displays the following output:

```
TPOINT=8003 DMPADR=47A8F7 USERID=PHAYES   COMP=SC   DATE=10/06/87 TIME=09:32:59
  MOD_REPORTS='DMS5CA  '
  L_VMCPLIST=HEXADECIMAL DUMP:
ADDR   OFFSET  DUMP DATA
038017 000000  06008700 00020A00 00000000 00000000  * ................ *
038027 000010  00000000 00000000 00000000 00000000  * ................ *
038037 000020  00000000 00000040                    * .......          *
```

Figure 50. Sample Output of the TRACE Subcommand (SFS Display)

Messages and Return Codes

AGWZTD440E
Operand missing or invalid

AGWZTD448E
Page *page* not found in dump

AGWZTF460E
Data field overlaps trailer record

AGWZTO461E
Formatted data entry exceeds maximum size

AGWZTR440E
Operand missing or invalid

AGWZTR441E
Conflicting operand: *operand*

AGWZTR449E
Non-numeric character in count; retry

AGWZTS447E
Trace table pointers invalid: Start = *start* End = *end* Current = *current*

AGWZTS448E
Page *page* not found in dump

AGWZTS450E
FROM location outside of trace table range: *fromloc* Start = *start* End = *end* Current = *current*

AGWZTS451E
FROM location is not a valid trace entry: *fromloc*

AGWZTS452E
Invalid trace entry found at *addr*

AGWZTS453E
Required resources are not available

AGWZTS454E
No trace entries found

AGWZTS455E
Attempted to go beyond storage boundary

AGWZTS457E
Trace entry search stopped at *addr1* To search to lower dump addresses, try address *addr2* To search to higher dump addresses, try addresses *addr3 addr4*

AGWZTS458E
Possible trace entry at *addr1* Use the FROM operand to display the entry

ATSZTF091E
DATA FIELD OVERLAPS TRAILER RECORD

ATSZTO076E
FORMATTED DATA ENTRY EXCEEDS MAXIMUM SIZE

ATSZTR075E
NON-NUMERIC COUNT CHARACTER - RETRY

ATSZTR077E
CONFLICTING OPERAND - *operand*

ATSZTR078E
OPERAND MISSING OR INVALID

ATSZTR087E
ATTEMPT TO GO BEYOND STORAGE BOUNDARY

ATSZTS079I
TRACE TABLE POINTERS INVALID: START = start END = end CURRENT = current

ATSZTS080I
"FROM" LOCATION OUTSIDE OF TRACE TABLE RANGE: *fromloc* START = start END = end CURRENT = current

ATSZTS081E
"FROM" LOCATION NOT A VALID TRACE ENTRY: *fromloc*

ATSZTS082E
INVALID TRACE ENTRY FOUND AT *addr*

ATSZTS083E
REQUIRED RESOURCES NOT AVAILABLE

ATSZTS084I
PAGE '*page*' NOT FOUND IN DUMP

ATSZTS086E
NO TRACE ENTRIES FOUND - *addr*

ATSZTS088E
UNABLE TO LOCATE TRACE TABLE POINTERS

ATSZTS092I
TRACE ENTRY SEARCH STOPPED AT *addr1* TO SEARCH TO LOWER DUMP ADDRESSES, TRY ADDRESS *addr2* TO SEARCH TO HIGHER DUMP ADDRESSES, TRY (ADDRESS *addr3* | "SCROLL")

ATSZTS093I
POSSIBLE TRACE ENTRY AT *addr* USE THE "FROM" OPERAND TO DISPLAY THE ENTRY

DMS5NB3950E
Non-numeric count character - Retry

DMS5ND3951E
Formatted data entry exceeds maximum size

DMS5NB3952E
Conflicting operand - *operand*

DMS5NB3953E
Operand missing or invalid

DMS5NC3954W
Trace table pointers invalid: Start = *start* End = *end* Current = *current*

DMS5NC3955W
"FROM" location outside of trace table range: *fromloc* Start = *start* End = *end* Current = *current*

DMS5NC3956E
"FROM" location not a valid trace entry: *fromloc*

DMS5NC3957E
Invalid trace entry found at *addr*

DMS5NC3958E
Required resources not available

DMS5NC3959W
Page *xxxxxxxx* not found in dump

DMS5NF3960E
Invalid trace point found in CPTRAP file

DMS5NC3961E
No trace entries found - *addr*

DMS5NC3962E
Attempt to go beyond storage boundary

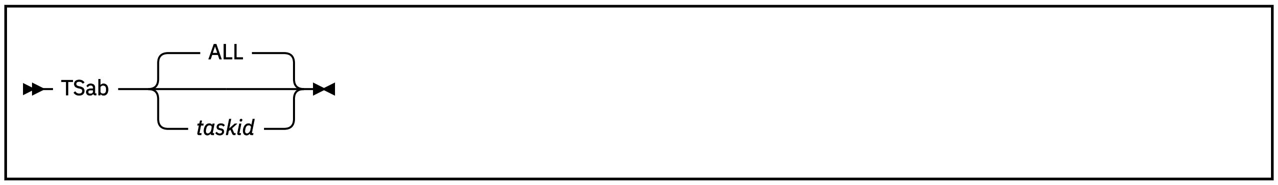
DMS5NC3963E
Unable to locate trace table pointers via *n*

DMS5NC3964I
Trace entry search stopped at *addr1* To search to lower dump addresses, try address *addr2* To search to higher dump addresses, try {address *addr3* | "SCROLL"}

DMS5NC3965I
Possible trace entry at *addr* Use the FROM operand to display the entry

DMS5NB3966E
IPCS TRACE subcommand missing or invalid

TSAB Subcommand (GCS,AVS,RSCS Dumps)



Purpose

Use the TSAB subcommand to display the subpool map and storage owned by a task.

Operands

taskid
identifies the task you want information about. The format is *nnnn*.

ALL
requests information for all tasks. ALL is the default.

Examples

The following is an example of the output of the TSAB subcommand. The first 32 bytes of the TSAB contain the 256 bit map of the subpools owned by the task.

```
TASK-ID  TASK-BLOCK  TASK-STORAGE-ANCHOR-BLOCK  CHAIN-HEADER

HHHH     HHHHHHHH    HHHHHHHH                    HHHHHHHH

SUBPOOL-MAP: (CONSISTING OF 64 HEX DIGITS)
```

Responses

Displays the:

- Task block address
- Task storage anchor block address
- Storage owned by the task
- 256-bit map of the subpools owned by the task.

Messages and Return Codes

GCTITL031S
Insufficient free storage is available

GCTITL504I
Page '*nnnnnnnn*' not found in dump

GCTIAL505I
TASKID '*xxxxxxxx*' invalid

GCTITL537I
Task block PRT is zero. Cannot find task load list

GCTITL538I
Task block list PRT is zero

GCTITA539I
NUCON extension PRT is zero. Cannot find task storage anchor block

GCTITA540I
TASKID table PRT is zero. Cannot find task storage anchor block

VMLOADL Subcommand (GCS,AVS,RSCS Dumps)

```
►►─ VMLoadl ─►◄
```

Purpose

Use the VMLOADL subcommand to display information about all programs loaded in this virtual machine.

Examples

The following is an example of the output of the VMLOADL subcommand.

```
MAJOR-NUCCBLK  MOD-NAME   MOD-ENTRY-ADDR  MOD-SIZE  MOD-ADDR

      HHHHHHHH EEEEEEEE   HHHHHHHH        HHHH      HHHHHHHH

MAJOR-NUCCBLK  ENTRY-NAME ENTRY-ADDRESS   TYPE

      HHHHHHHH EEEEEEEE   HHHHHHHH        EEEEEEEE
        .         .         .                .
        .         .         .                .
        .         .         .                .
```

Responses

Displays for each module loaded in this virtual machine:

- Address of the control block containing related information
- Associated program name
- Program address
- Program size
- Entry point address.

For an ALIAS or IDENTIFY-specified entry point, this subcommand displays:

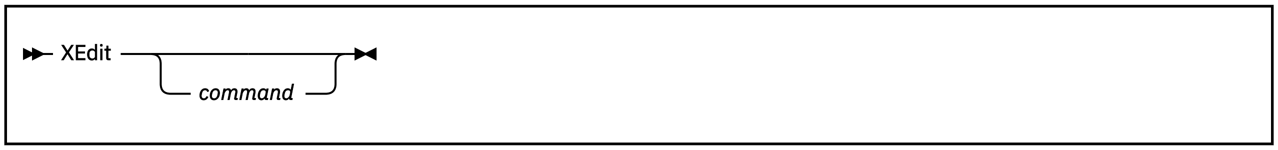
- Address of the control block containing related information
- Entry point name
- Entry point address
- Type of control block (ALIAS or IDENTIFY).

Messages and Return Codes

GCTIVL504I
Page '*nnnnnnnn*' not found in dump

GCTIVL533I
The virtual machine load list is empty

XEDIT Subcommand



Purpose

Use the XEDIT subcommand to force the Dump Viewing Facility to pass the command line to XEDIT for execution.

Operands

command
is any valid XEDIT subcommand or macro and its operands.

Usage Notes

1. Any XEDIT subcommand should be prefaced with XEDIT to prevent the Dump Viewing Facility from processing the subcommand.

Appendix A. Using Attachment Interfaces

The following attachment interfaces are provided by the Dump Viewing Facility:

- The MAP attachment interface
- Exit routine interfaces
- Block tables.

An attachment interface is defined as the logical interconnection and interaction between or to software programs that enable the programs to function together.

MAP Attachment Interface

The Dump Viewing Facility MAP command enables you to compress load map(s) to create a module map. The module map can then be appended to a dump by using the ADDMAP command. The format of the generated module map, as used by the ADDMAP command, is described in Appendix D, "Module Map Architecture (Used by ADDMAP)," on page 189. In addition to the load map(s), the Dump Viewing Facility needs specific information in order to create a module map. This information, for the supported dump types, resides in the HCSTAB ASSEMBLE file. Entries can be added to this file, or existing entries can be modified by using the TABENTRY utility macro. This procedure for modifying the HCSTAB ASSEMBLE file is described under "Modifying the HCSTAB Table" on page 139. The description and specification information for the TABENTRY utility macro is described in Appendix C, "Dump Viewing Facility Utilities," on page 163.

HCSTAB Table

The HCSTAB table contains the map information required by the MAP command for creating a module map from a load map(s). The entries in the table are in a specified order; one entry may depend on another entry. For example, the CMS entry must precede the TSAF entry in the table.

HCSTAB Table Format

The table is organized by dump types supported by the Dump Viewing Facility as follows:

- CMS
- GCS/XA
- AVS
- PVM
- RSCS
- SFS (including CRR)
- TSAF.

Modifying the HCSTAB Table

The TABENTRY utility macro is provided to create a map information entry in the HCSTAB table for a specific dump type. This entry provides the information required for the compression of the dump's associated load map(s) which then can be used to generate a module map for the dump. The description and specification information for the TABENTRY macro is described in Appendix C, "Dump Viewing Facility Utilities," on page 163. Use the following procedure for modifying the HCSTAB table:

1. Place the HCSTAB ASSEMBLE file on your A-disk or other writable disk.
2. Use the TABENTRY utility macro to add or modify a table entry.
3. Assemble the HCSTAB ASSEMBLE file.

4. Correct any problems identified by error messages.
5. Place the HCSTAB TEXT file on your A-disk.
6. Use the VMSES/E Local Modification procedure to regenerate the MAP module, see the *z/VM: Service Guide*.

Exit Routine Interfaces

PI

The Dump Viewing Facility provides several types of installation wide exit interfaces for virtual machine dumps:

- Extraction routines
- Formatting routines called from DUMPSCAN
- Formatting routines called from PRTDUMP

The routine names for the supported dump types are located in the Dump Viewing Facility HCSTBL ASSEMBLE file. Entries can be added to this file, or existing entries can be modified. See “Modifying the HCSTBL Table” on page 144 for the procedure for adding or modifying table entries.

These routines can use the services that the Dump Viewing Facility provides as described in Appendix B, “Dump Viewing Facility Services,” on page 153, under “SVC 199 Services” on page 153 and “Miscellaneous Services” on page 159.

Extraction Routines

The Dump Viewing Facility sets up this exit to allow the extraction routine for a specific virtual machine dump to gain control, extract the required information, and optionally return the information to Dump Viewing Facility through use of SVC 199 services. Prerequisites for calling the extraction routines are the following:

- The extraction routine must be accessible
- The virtual machine dump, which is the object of DUMPSCAN or PRTDUMP, must reside on a writable disk.

If these prerequisites are satisfied, the extraction routine is invoked during the initialization phase of either the DUMPSCAN or PRTDUMP command. If the extraction routine is successful, it is invoked only the first time the dump is viewed or printed by DUMPSCAN or PRTDUMP, respectively. If unsuccessful, the extraction routine will be called again on the next invocation of either DUMPSCAN or PRTDUMP.

Extraction Routine Interface

The extraction routine must provide proper entry and exit linkage. When the extraction routine receives control, the registers contain the following:

Register	Contents
0	Not specified
1	The address of the 8-byte return string, initially blank (filled with X'40404040')
2-12	Not specified
13	The savearea address
14	The return address

15
The entry point address

On return from the extraction routine, the Dump Viewing Facility interprets the return string as follows:

Return String Values
Dump Viewing Facility Actions

'ERROR '
The extraction routine was unsuccessful. If SVC 199 codes 10, 20, or 80 were issued by the extraction routine, the services for these codes are not performed. If the dump is viewed using DUMPSCAN or printed by PRTDUMP again, the extraction routine is called.

All others
The extraction routine was successful. If SVC 199 codes 10, 20, or 80 were issued by the extraction routine, the services for these codes will be performed. A flag is set in the dump's symptom record informing Dump Viewing Facility that the extraction routine for this dump has already been performed. Therefore it is not called on subsequent invocations of either the DUMPSCAN or PRTDUMP commands.

Note: On return from the extraction routine, Dump Viewing Facility does not check the return code.

Formatting Routines Called from DUMPSCAN

The Dump Viewing Facility sets up this exit to allow the formatting routine for a specific virtual machine dump type to gain control during a DUMPSCAN session in order to format data areas.

The formatting routines are invoked during a Dump Viewing Facility DUMPSCAN session. This occurs when DUMPSCAN does not handle a subcommand. DUMPSCAN will not handle a subcommand in the following situations:

- The subcommand is not recognized by DUMPSCAN
- The subcommand is the TRACE subcommand and the dump being viewed is a virtual machine dump
- The subcommand is a scrolling subcommand (FORWARD, BACKWARD, SCROLL, SCROLLU, SCROLL U, or NULLLINE) and the special scrolling interface was established previously by a formatting routine invoked by another subcommand (for example, the TRACE subcommand). See “Special Scrolling Interface” on page 143 for additional information.

  Note: Special scrolling is disabled after a user issues a successful Dump Viewing Facility subcommand that can be scrolled.

Formatting Routine Interface for Nonscrolling Subcommands

The formatting routine must provide proper entry and exit linkage. If the subcommand was a nonscrolling subcommand, the registers will contain the following when the formatting routine receives control :

Register
Contents

0-1
Not specified

2
The address of the tokenized input list (tokens are 8 characters in length and converted to uppercase; the token list is ended by X'FFFFFFFFFFFFFFFF')

3
The address of the untokenized input list (the input list is converted to uppercase and is ended by X'FFFFFFFFFFFFFFFF')

4-12
Not specified

13
The savearea address

14
The return address

15
The entry point address

On return from the formatting routine, the Dump Viewing Facility handles the return code as follows:

Return Codes
Dump Viewing Facility Actions

0 or 4
The formatting routine processed the subcommand successfully, and DUMPSCAN prompts for a new subcommand.

8 or 12
The formatting routine was unsuccessful in processing the recognized subcommand. DUMPSCAN issues the Dump Viewing Facility message 270I, then prompts for new subcommand.

16
The formatting routine did not recognize the subcommand. DUMPSCAN passes the subcommand to XEDIT for processing.

All others
Reserved—currently processed as follows:

Formatting routine was unsuccessful in processing the recognized subcommand. DUMPSCAN issues Message 270I and prompts for new subcommand.

Note: If a formatting routine does not recognize a subcommand, it **must** return a return code of 16. This return code instructs DUMPSCAN to pass the subcommand to other environments (XEDIT, CMS, and CP) for potential processing.

Formatting Routine Interface for Scrolling Subcommands

The formatting routine must provide proper entry and exit linkage. If the subcommand is a scrolling subcommand, the registers contain the following when the formatting routine receives control:

Register
Contents

0-1
Not specified

2
The address of the second token in the tokenized input list (tokens are 8 characters in length and converted to uppercase; token list is ended by X'FFFFFFFFFFFFFFFF')

3
The address of the first argument of the untokenized input list (input list is converted to uppercase and is ended by X'FFFFFFFFFFFFFFFF')

4-12
Not specified

13
The savearea address

14
The return address

15
The entry point address

Note: If the scrolling subcommand has no second token, registers 2 and 3 point to X'FFFFFFFFFFFFFFFF'. If the subcommand itself consists of two tokens (SCROLL U), and there is no third token, registers 2 and 3 point to X'FFFFFFFFFFFFFFFF'. If the subcommand was NULLLINE, both registers 2 and 3 point to X'FFFFFFFFFFFFFFFF'.

On return from the formatting routine, the Dump Viewing Facility handles the return code as follows:

Return Codes	**Dump Viewing Facility Action**
0 or 4	The formatting routine processed the subcommand successfully, and DUMPSCAN prompts for a new subcommand.
8 or 12	The formatting routine was unsuccessful in processing the recognized subcommand. DUMPSCAN issues message 270I and prompts for a new subcommand.
16	The formatting routine did not recognize the subcommand. DUMPSCAN passes the subcommand to XEDIT for processing.
All others	Reserved—currently being processed as follows: The formatting routine was unsuccessful in processing the recognized subcommand. DUMPSCAN issues message 270I and prompts for a new subcommand.

Note: If a formatting routine does not recognize a subcommand, it **must** return a return code of 16. This return code instructs DUMPSCAN to pass the subcommand to other environments (XEDIT, CMS, and CP) for potential processing.

Special Scrolling Interface

The special scrolling interface supports scrolling by user exits such as the TSAF Trace command. The following external variables are supported by DUMPSCAN:

Name	Length	Contents
HEXAD	Fullword	Contains the address of the first trace entry in a previous display
SCROLLEN	Fullword	Contains the address of the last trace entry in a previous display
FEDFEXSW	One byte	Contains the scroll switch *S*
REUSEAD	Fullword	Contains the address of the routine for the REUSE subcommand
SCROLAD	Fullword	Contains the address of the routine for the SCROLL subcommand
SCROLUAD	Fullword	Contains the address of the routine for the SCROLLUP subcommand
PRINTONE	One byte	X'0F' indicates output of previous subcommand should be displayed.

When the Dump Viewing Facility invokes the format routine, register 2 points to the next parameter in the tokenized command parameter list.

Formatting Routines Called from PRTDUMP

The Dump Viewing Facility sets up this exit to allow the formatting routine for a specific virtual machine dump type to gain control in order to format data areas for printing. This formatting routine is invoked through the Dump Viewing Facility PRTDUMP command.

Formatting Routine Interface

The formatting routine must provide proper entry and exit linkage. When the formatting routine receives control, the registers contain:

Registers	**Contents**
0	Not specified
1	The address of the tokenized list of PRTDUMP options that were specified on the PRTDUMP command line but were not recognized by PRTDUMP (tokens are 8 characters in length, converted to uppercase, and the token list is ended by X'FFFFFFFFFFFFFFFF')
2-12	Not specified
13	The save area address
14	The return address
15	The entry point address

Note: The tokenized options list (pointed to by register 1) consists of a maximum of nineteen 8-character tokens. Options greater than 8 characters in length are truncated to 8 characters.

On return from the formatting routine, PRTDUMP will not check the return code.

HCSTBL Table

The Dump Viewing Facility provides the capability for exit routines to extract, view, and print virtual machine dump data. It uses a table, HCSTBL, as the attachment interface for this function. This table is contained within a file named HCSTBL ASSEMBLE and is shipped with the Dump Viewing Facility. It contains information associated with each supported virtual machine dump type.

HCSTBL Table Format

The format of HCSTBL supported by the Dump Viewing Facility is as follows:

- CMS
- GCS/XA
- AVS
- PVM
- RSCS
- SFS (including CRR)
- TSAF.

Modifying the HCSTBL Table

The TBLENTRY utility macro is provided for creating an entry in the HCSTBL table for a specific dump type. This entry provides the necessary extraction and formatting information for a specific dump type. The description and specification information for this utility macro is described in Appendix C, “Dump Viewing Facility Utilities,” on page 163. Use the following procedure for modifying the HCSTBL table:

1. Place the HCSTBL ASSEMBLE file on your A disk or other writable disk.
2. Use the TBLENTRY utility macro to add or modify a table entry.
3. Assemble the HCSTBL ASSEMBLE file.

4. Correct any problems identified by error messages.
5. Place the HCSTBL TEXT file on your A disk.
6. Use the VMSES/E Local Modification procedure to regenerate the DUMPSCAN and PRTDUMP modules, see the *z/VM: Service Guide*.

PI end

Block Table Architecture

The BLOCK subcommand displays a control block by mapping the contents of storage into a predefined format. This is accomplished by using a previously defined description of the control block called the block definition file. The block definition file is listed in a control file which is named in HCSTBL.

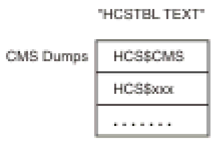


Figure 51. DUMPSCAN BLOCK Tables Diagram

HCSTBL is shipped with entries for each of the Dump Viewing Facility supported dump types.

Creating Block Definition Files

You can alter the block definitions and create new definitions in user block table files. You must observe the following limitations, however:

- A block table file must have a logical record size of 80
- The record format must be fixed
- The file cannot exceed 32,656 records in length
- A maximum of 2048 blocks can be defined per block table
- The file name of the file must be unique. This is necessary to ensure the right block definitions are loaded into BLOCK during initialization. If the name is not unique, BLOCK loads the first occurrence of the name based on the CMS minidisk search order.

Creating BLOCK Control Files

After the BLOCK table file is built, you must determine in which of the BLOCK control files the new definition file name should reside, and what order position the file will occupy.

A maximum of four block table files can be used in a single block session.

The block control file is a file that tells BLOCK which block table files to load during initialization. The definition files loaded depend on the type of dump the user is currently examining. When BLOCK initialization starts, BLOCK uses the dump type to search a table called HCSTBL. See "HCSTBL Table" on page 144 for more information on HCSTBL. When a match is found on the DUMPTYPE, the entry in the HCSTBL is checked for a block control file. If that exists, the suffix is appended to the characters HCS$ to form the file name of the control file name (the file type is always TABLE) for that dump type. BLOCK then searches for that control file and extracts the names of the block definition files for the dump. After you know the name of the control file for the dump being examined, you simply put the name of the new file in the order position desired. The position of the file is important. BLOCK searches for control block names sequentially. This means that the first definition file is searched first, the second searched second, and so on. After the BLOCK subcommand finds a match to the name entered on BLOCK invocation, the file search stops, and BLOCK maps the data based on that definition. BLOCK thus does not recognize duplicate names within a definition file.

After you have added the new definition file names, the file should be saved on a minidisk that is ahead of the original control file disk in the CMS search order.

The following is an example of a simple BLOCK control file. The file has records 80 bytes long.

```
*************************************************************
* The base CMS control blocks for z/VM                      *
*************************************************************
CMSBLOCK CBMAP
```

In this example, MYFILE CBMAP is added to a BLOCK control file:

```
*************************************************************
* The base CMS control blocks for z/VM                      *
*************************************************************
MYFILE CBMAP
CPMBLOCK CBMAP
```

Adding Block Definition Files

Assume you have an updated CMS control block source copy file which you want to define to DUMPSCAN such that the BLOCK subcommand can map dump data for this control block using the updated version. For your example, you have a copy file for the SAVBK control block, "DMSABN COPY" located on your A-disk.

1. Invoke BLOCKDEF utility command to generate a block definition file containing formatted DSECT for the SAVBK. Suppose you wish to name this new block definition file, "MY CBMAP".

   "BLOCKDEF DMSABN COPY A MY CBMAP A (COPY".

   The optional parameter COPY tells BLOCKDEF the input file name is the name of a single control block to be processed. See "BLOCKDEF Utility Command" in Appendix C for a description of all the optional parameters used with BLOCKDEF.
2. Locate the "HCS$xxx TABLE" control file which is appropriate for the type of dump you are processing.

   In your example, you are processing a CMS control block, so you want to select the control file for CMS dumps, "HCS$CMS TABLE". These control files are named in HCSTBL, which can be updated. "HCS$CMS TABLE" is supplied with the product. See “Modifying the HCSTAB Table” on page 139 for a description on how to update HCSTBL.
3. Add the name of the new block definition file created with BLOCKDEF above, "MY CBMAP", to the "HCS$CMS TABLE" control file.

   Up to 4 block definition files can be listed in one control file.
4. You are now ready to use your new block definition file with the BLOCK subcommand.

   Note: If you were already in a DUMPSCAN session when you created the new block definition file for SAVBK, you MUST exit the DUMPSCAN session and start again, before the change will take effect. DUMPSCAN only initializes the control block mappings for a particular dump type once during initialization time.

Block Definitions

The control block descriptions are formatted as follows:

- Each field within a control block is described to the BLOCK subcommand in a single record called a block definition record (see Figure 52 on page 146).
- The entire control block description is made up of groups of these definition records and is called a block definition (see Figure 53 on page 147).
- The block definitions are stored in large files and are called block table files (see Figure 54 on page 147).

```
030       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                                IDENTIFIER
```

Figure 52. A Block Definition Record: A Single Field within a Control Block

Note: See “Block Descriptor Record Format” on page 147 for record layout.

```
*** USER BLOCK NONDISPLAYABLE COMMENT
000       0    (0) STRUCTURE    29  USERBLOK
010       0    (0) BIT           1  USERFLGA       EVENT STATUS FLAGS
011            1...  ....           USERBIT1       I/O IN PROGRESS
012            .1..  ....           USERBIT2       DEACTIVATE STARTED
012            ..1.  ....           USERBIT3       SESSION ENDED
012            ...1  ....           USERBIT4       PURGE Q REQUESTED
012            ....  1...           USERBIT5       PURGE Q COMPLETED
012            ....  .1..           USERBIT6       DEACTIVATE COMPLETE
012            ....  ..1.           USERBIT7       I/O COMPLETE
013            ....  ...1           USERBIT8       PURGE Q I/O WAIT
010       1    (1) BIT           1  USERFLGB       EVENT STATUS FLAGS #2
020       2    (2) FIXED(U)      2  *              RESERVED
040       4    (4) POINTER       4  USERLINK       LINK POINTER
030       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                                IDENTIFIER
060      12    (C) OFFSET        4  USERREGF       SAVED RETURN CODE
0A0                                                FROM CALL
080      16   (10) FIXED(S)      4  USERREGE       SAVED REGE FROM
0A0                                                PRIOR CALL
090      20   (14) AREA          8  USERTIME       TIME OF DISPATCH TO
0A0                                                CPU 1 DISPATCHER/
0A0                                                SCHEDULER ROUTINE
010      28   (1C) BIT           1  USERFLGC       LOCK FLAGS
011            1..1  .11.           USERLOC2       DISPATCH STATUS FLAGS
0A1                                                FOR THE PRIMARY CPU
0A1                                                DISPATCHER
```

Figure 53. A Block Definition: The Group of Records Defining a Single Block

```
*** FIRST BLOCK DEFINITION
000       0    (0) STRUCTURE    29  USERBLOK
010       0    (0) BIT           1  USERFLGA       EVENT STATUS FLAGS
011            1...  ....           USERBIT1       I/O IN PROGRESS
012            .1..  ....           USERBIT2       DEACTIVATE STARTED
012            ..1.  ....           USERBIT3       SESSION ENDED
013            ...1  ....           USERBIT4       PURGE Q REQUESTED
010       1    (1) BIT           1  USERFLGB       EVENT STATUS FLAGS #2
020       2    (2) FIXED(U)      2  *              RESERVED
040       4    (4) POINTER       4  USERLINK       LINK POINTER
030       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                                IDENTIFIER
060      12    (C) OFFSET        4  USERREGF       SAVED RETURN CODE
0A0                                                FROM CALL
080      16   (10) FIXED(S)      4  USERREGE       SAVED REGE FROM
0A0                                                PRIOR CALL
090      20   (14) AREA          8  USERTIME       TIME OF DISPATCH TO
0A0                                                THE CPU 1 DISPATCHER/
0A0                                                SCHEDULER ROUTINE
010      28   (1C) BIT           1  USERFLGC       LOCK FLAGS
011            1..1  .11.           USERLOC2       DISPATCH STATUS FLAGS
0A1                                                FOR THE PRIMARY CPU
0A1                                                DISPATCHER
*** SECOND BLOCK DEFINITION
000       0    (0) STRUCTURE    34  BLOKBLOK
030       0    (0) CHARACTER     4  BLOKCBID       THE BLOCKS' ID FIELD
040       4    (4) POINTER       4  BLOKLINK       LINK POINTER
040       8    (8) POINTER       4  BLOKSTAT       ACTIVATE STATUS
040      12    (C) POINTER       4  BLOKWAIT       WAIT EVENT BLOCK
0A0                                                POINTER
090      24   (18) AREA          8  BLOKTIME       TIME OF DISPATCH
010      32   (20) BIT           1  BLOKWFLG       WAIT FLAGS
010      33   (21) BIT           1  *              RESERVED
```

Figure 54. A Block Table File Containing Multiple Block Definitions

Block Descriptor Record Format

You can build a block definition record by using the format shown in Figure 55 on page 147.

```
040       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                                IDENTIFIER
```

Figure 55. The Block Descriptor Record

Where:

Table 10. Block Descriptor Record Format

Columns	Field Name	Field Description
1-3	Key field	A 3-digit value that provides information about the field to BLOCK. The digits are used as follows: **Column 1** Default character field **Column 2-3** Data type indicator The possible combinations are: **00** STRUCTURE—Start of definition **10** BIT—BIT data **11** BIT—Bit subrecord start (on byte boundary) **12** BIT—Bit subrecord **13** BIT—Bit subrecord end **20** FIXED(U)—Fixed unsigned data **30** CHARACTER—Character data **40** POINTER—Pointer data **50** Reserved for future IBM use **60** OFFSET—Offset data **70** Reserved for future IBM use **80** FIXED(S)—Fixed signed data **90** AREA—Area data **A0** Independent record comment **A1** Independent subrecord comment ****** Nondisplayable comment
7-11	Decimal offset	The field's offset within the block. This field is right-justified.
13-18	Hexadecimal offset	The field's offset within the block. This field is right-justified and bracketed by parentheses.
20-28	Data type	The type of data contained in the field

Table 10. Block Descriptor Record Format (continued)		
Columns	**Field Name**	**Field Description**
30-34	Field/block size	The decimal size of the field in bytes. This field is right-justified.
37-50	Field/block name	The name of the field. The name field can also contain an array element count value. This count must be bracketed by parentheses; for example, TESTDATA(1024). This indicates that TESTDATA is an array of 1024 elements.
52-79	Field comments	The comments describing the field

Note: Columns not specifically assigned to a field must contain blanks.

Block Descriptor Record – BIT Subrecord Format

There is an alternate format for bit subrecord format records. Figure 56 on page 149 shows an example:

```
011           1...  ....          USERBIT1       I/O IN PROGRESS
```

Figure 56. Alternate Format for Bit Subrecord

Where:

Table 11. Block Descriptor Record–BIT Subrecord Format		
Columns	**Field Name**	**Field Description**
01-03	Key field	A 3-digit value that provides information about the field to BLOCK. The digits are used as follows: **Column 1** Default character field **Column 2-3** Data type indicator The possible combinations are: **10** BIT—BIT data **11** BIT—Bit subrecord start (on byte boundary) **12** BIT—Bit subrecord **13** BIT—Bit subrecord end
16-25	Bit map field	The bit placement map used to format bit display
37-50	Field/block name	The name of the bits
52-79	Field comments	The comments describing the field

Note: Columns not specifically assigned to a field must contain blanks.

Default Display Fields

You can get a display of the default display fields by omitting the ALL and PROMPT options on the BLOCK subcommand. You select the fields to be displayed as default fields. This is accomplished by specifying a default field indicator in the definition header record of a block definition as shown in Figure 57 on page 150.

```
000       0    (0) STRUCTURE    29  USERBLOK
```

Figure 57. Block Header Record with No Default Settings

Note: If you want to use a nondisplayable comment, you cannot use an asterisk for the default character within that block.

The default character for a block is set by altering the first character of the KEY field. For example, we will use the letter D in Figure 58 on page 150 to signify that a field is a default field. The user may specify any valid EBCDIC character as the default character.

```
D00       0    (0) STRUCTURE    29  USERBLOK
```

Figure 58. Block Header Record Set to Default of D

Altering a block definition header record KEY field indicates to the BLOCK subcommand that any field record within the definition that has the same character in the first character position of its own KEY field is to be considered a default field. For example, the block field shown in Figure 59 on page 150 would be considered a default field.

```
D30       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
DA0                                                IDENTIFIER
```

Figure 59. Field Record Set to Default of D

Tailoring a Block Table File

You can customize an existing definition by:

- Changing the control block name in the header record
- Changing the names of various fields
- Deleting fields of no interest
- Adding new fields, such as bit subrecords
- Changing comments for the field.

Use caution when modifying the following fields in the definition:

- Control block and field size values. The field size value tells BLOCK how much data to map into that particular field. The block size value tells BLOCK how much data to get from the dump to map into the block.
- Field offset values. These fields tell BLOCK exactly where, within the storage of the control block, the data to be mapped is located.

In the following record, the control block name is changed from:

```
D00       0    (0) STRUCTURE    29  USERBLOK
```

to:

```
D00       0    (0) STRUCTURE    29  MYBLOCK1
```

In the following record, the field name and the comment are changed from:

```
D30       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
DA0                                                IDENTIFIER
```

to:

```
D40      8    (8) CHARACTER     4  BLOCKID        THE BLOCKS'
DA0                                               ID FIELD
```

In the following record:

```
000      0    (0) STRUCTURE    29  USERBLOK
010      0    (0) BIT           1  USERFLGA       EVENT STATUS FLAGS
011           1...  ....           USERBIT1       I/O IN PROGRESS
012           .1..  ....           USERBIT2       DEACTIVATE STARTED
012           ..1.  ....           USERBIT3       SESSION ENDED
013           ...1  ....           USERBIT4       PURGE Q REQUESTED
010      1    (1) BIT           1  USERFLGB       EVENT STATUS FLAGS #2
020      2    (2) FIXED(U)      2  *              RESERVED
040      4    (4) POINTER       4  USERLINK       LINK POINTER
030      8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                               IDENTIFIER
060     12    (C) OFFSET        4  USERREGF       SAVED RETURN CODE
0A0                                               FROM CALL
080     16   (10) FIXED(S)      4  USERREGE       SAVED REGE FROM
0A0                                               PRIOR CALL
090     20   (14) AREA          8  USERTIME       TIME OF DISPATCH TO
0A0                                               THE CPU 1 DISPATCHER/
0A0                                               SCHEDULER ROUTINE
010     28   (1C) BIT           1  USERFLGC       LOCK FLAGS
011           1..1  .11.           USERLOC2       DISPATCH STATUS FLAGS
0A1                                               FOR THE PRIMARY CPU
0A1                                               DISPATCHER
```

a bit breakdown is added to a definition:

```
000      0    (0) STRUCTURE    29  USERBLOK
010      0    (0) BIT           1  USERFLGA       EVENT STATUS FLAGS
011           1...  ....           USERBIT1       I/O IN PROGRESS
012           .1..  ....           USERBIT2       DEACTIVATE STARTED
012           ..1.  ....           USERBIT3       SESSION ENDED
012           ...1  ....           USERBIT4       PURGE Q REQUESTED
012           ....  1...           USERBIT5       PURGE Q COMPLETED
012           ....  .1..           USERBIT6       DEACTIVATE COMPLETE
012           ....  ..1.           USERBIT7       I/O COMPLETE
013           ....  ...1           USERBIT8       PURGE Q I/O WAIT
010      1    (1) BIT           1  USERFLGB       EVENT STATUS FLAGS #2
020      2    (2) FIXED(U)      2  *              RESERVED
040      4    (4) POINTER       4  USERLINK       LINK POINTER
030      8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                               IDENTIFIER
060     12    (C) OFFSET        4  USERREGF       SAVED RETURN CODE
0A0                                               FROM CALL
080     16   (10) FIXED(S)      4  USERREGE       SAVED REGE FROM
0A0                                               PRIOR CALL
090     20   (14) AREA          8  USERTIME       TIME OF DISPATCH TO
0A0                                               THE CPU 1 DISPATCHER/
0A0                                               SCHEDULER ROUTINE
010     28   (1C) BIT           1  USERFLGC       LOCK FLAGS
011           1..1  .11.           USERLOC2       DISPATCH STATUS FLAGS
0A1                                               FOR THE PRIMARY CPU
0A1                                               DISPATCHER
```

In this record:

```
000       0    (0) STRUCTURE    29  USERBLOK
010       0    (0) BIT           1  USERFLGA       EVENT STATUS FLAGS
011            1...  ....           USERBIT1       I/O IN PROGRESS
012            .1..  ....           USERBIT2       DEACTIVATE STARTED
012            ..1.  ....           USERBIT3       SESSION ENDED
012            ...1  ....           USERBIT4       PURGE Q REQUESTED
012            ....  1...           USERBIT5       PURGE Q COMPLETED
012            ....  .1..           USERBIT6       DEACTIVATE COMPLETE
012            ....  ..1.           USERBIT7       I/O COMPLETE
013            ....  ...1           USERBIT8       PURGE Q I/O WAIT
010       1    (1) BIT           1  USERFLGB       EVENT STATUS FLAGS #2
020       2    (2) FIXED(U)      2  *              RESERVED
040       4    (4) POINTER       4  USERLINK       LINK POINTER
030       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                                IDENTIFIER
060      12    (C) OFFSET        4  USERREGF       SAVED RETURN CODE
0A0                                                FROM CALL
080      16   (10) FIXED(S)      4  USERREGE       SAVED REGE FROM
0A0                                                PRIOR CALL
090      20   (14) AREA          8  USERTIME       TIME OF DISPATCH TO
0A0                                                THE CPU 1 DISPATCHER/
0A0                                                SCHEDULER ROUTINE
```

USERBIT5 through USERBIT8 are deleted:

```
000       0    (0) STRUCTURE    29  USERBLOK
010       0    (0) BIT           1  USERFLGA       EVENT STATUS FLAGS
011            1...  ....           USERBIT1       I/O IN PROGRESS
012            .1..  ....           USERBIT2       DEACTIVATE STARTED
012            ..1.  ....           USERBIT3       SESSION ENDED
013            ...1  ....           USERBIT4       PURGE Q REQUESTED
010       1    (1) BIT           1  USERFLGB       EVENT STATUS FLAGS #2
020       2    (2) FIXED(U)      2  *              RESERVED
040       4    (4) POINTER       4  USERLINK       LINK POINTER
030       8    (8) CHARACTER     4  USERCBID       CONTROL BLOCK
0A0                                                IDENTIFIER
060      12    (C) OFFSET        4  USERREGF       SAVED RETURN CODE
0A0                                                FROM CALL
080      16   (10) FIXED(S)      4  USERREGE       SAVED REGE FROM
0A0                                                PRIOR CALL
090      20   (14) AREA          8  USERTIME       TIME OF DISPATCH TO
0A0                                                THE CPU 1 DISPATCHER/
0A0                                                SCHEDULER ROUTINE
```

Appendix B. Dump Viewing Facility Services

PI

The following services are provided by the Dump Viewing Facility:

- SVC 199 services
- Miscellaneous services.

SVC 199 Services

The SVC 199 type of communication provides the interface between the Dump Viewing Facility and the exit routines. These services are used by:

- Extraction routines
- Formatting routines called by DUMPSCAN
- Formatting routines called by PRTDUMP.

Note: All codes are available to the DUMPSCAN command unless otherwise noted.

Following are the codes that the Dump Viewing Facility supports:

- **Code = 10**: Send the keyword symptom data to the Dump Viewing Facility for inclusion in the SYMPTOM record.

  ```
  PLIST     DS        0F
            DC        AL4(*-*)    The address of the keyword list
            DC        H'10'       The code field
            DC        H'0'        The number of entries in the list
  ```

  The keyword list must comply with the following rules:

  – All data must be presented in printable format

  – The total number of keywords must not exceed 15

  – The minimum number of keywords is four

  – The total length of keyword plus data, including a suffix of five blanks, is 20 characters. The first 15 characters are used.

  Return codes for register 15 are:

  Return Code

  Explanation

  0

  Successful operation

  4

  An invalid number of entries (<4 or >15)

- **Code = 20**: Send additional data to the Dump Viewing Facility for inclusion in the SYMPTOM record.

  ```
  PLIST     DS        0F
            DC        AL4(*-*)    The address of the data list
            DC        H'20'       The code field
            DC        H'0'        The number of 80-bytes entries in the list
  ```

  The data list must comply with the following rules:

  – All data is presented in printable format

  – Entries are 80 bytes long, including spaces

- Total number of entries do not exceed 15.

Return codes for register 15 are:

Return Code
Explanation

0
Successful operation

4
An invalid number of entries (>15)

- **Code = 30**: Request a work buffer

```
PLIST    DS      0F
         DC      AL4C(*-*)   Address of buffer returned to caller
         DC      H'30'       The code field
         DC      H'0'        The number of buffers requested
```

Note: The caller needs a work buffer. Up to six 4 KB buffers may be requested. The request is denied if all space asked for cannot be provided. Buffers are on page boundaries and are contiguous.

Return codes for register 15 are:

Return Code
Explanation

0
Successful operation

4
Insufficient storage

8
Invalid request

- **Code = 31**: Free a work buffer

```
PLIST    DS      0F
         DC      AL4(*-*)    The address of the buffer(s) to be returned
         DC      H'31'       The code field
         DC      H'0'        The number of 4 KB buffers to be returned
```

Frees storage previously obtained with SVC subcode 30.

Return codes for register 15 are:

Return Code
Explanation

0
Successful operation

4
Address invalid

8
Invalid request

- **Code = 40**: Request data from an address

```
PLIST    DS      0F
         DC      AL4(*-*)    The address of the data wanted
         DC      H'40'       The code field
         DC      H'0'        The number of bytes read contiguous to the
                             address requested and the end of a 12 KB buffer
         DC      AL4(*-*)    The address of the 12 KB buffer returned to the
                             caller that contains the address of the desired
                             data
```

The address of the data requested is the first entry in the buffer returned. The buffer varies in length if the next page of the dump was not dumped to the page in which the address requested was found. The last halfword of the PLIST contains the total number of consecutive bytes (a maximum of 12 KB).

Return codes for register 15 are:

Return Code
  Explanation

0
  Successful operation

4
  Page not in dump

100
  Read error

Note:

1. The next subcode call (40 or 41) overlays the buffer returned by the previous invocation of subcode 40 or 41.
2. When the requested address exceeds dump storage size, register 15 is set to 4, and the address of the 12 KB buffer returned is set to X'000000FF'.

- **Code = 41**: Request data from an address

```
PLIST    DS       0F
         DC       AL4(*-*)    The address of the data wanted
         DC       H'41'       The code field
         DC       H'0'        The number of usable bytes returned to the user.
                              This count varies.
         DC       AL4(*-*)    The address of the buffer returned to the caller
                              containing the address of the desired data. The
                              address requested is rounded down to a page
                              boundary.
```

The address in the buffer points to the beginning of the page containing the address of the requested data. The preceding page and the following page may also be present. The purpose is to provide the page before and the page after the requested page. The user must index into the page for his address or use SVC 199 code 40. The number of usable bytes (fourth PLIST field) contains the total number of bytes (a maximum of 12 KB).

Return codes for register 15 are:

Return Code
  Explanation

0
  Successful operation

1
  Preceding page not present

2
  Following page not present

3
  Preceding and following pages not present

4
  Page not in dump

100
  Read error

Note:

1. The next subcode call (40 or 41) overlays the buffer returned by the previous invocation of subcode 40 or 41.

2. When the requested address exceeds dump storage size, register 15 is set to 4, and the address of the 12 KB buffer returned is set to X'000000FF'.

- **Code = 50**: Request the dump file information record (DFIR). This record contains the general purpose registers, PSW, and dump ID storage at the time the dump was taken.

  These three fields will be at the offsets specified in the DMPINREC format.

  Table 12. DMPINREC Control Block

Field Description	Length	Offset
General Purpose Registers	64 bytes	X'0'
PSW	8 bytes	X'3D8'
Dump ID	100 bytes	X'434'

  Note: This subcode is intended for use by exit routines that have been coded to be used with the VM/370 interactive problem control system (IPCS), and that are migrating to the VM Dump Viewing Facility. Any exit routines written specifically for use with the Dump Viewing Facility should use subcode 51.

```
PLIST    DS       0F
         DC       AL4(*-*)    Address of 4 KB buffer to be used to return the
                              DMPINREC
         DC       H'50'       The code field
```

  This record may not have any value for a virtual machine dump. The value of this record should be determined by the user. All fields should be referenced using their DSECT names.

  Return codes for register 15 are:

  Return Code
  Explanation

  0
  Successful operation

  4
  Internal error

- **Code = 51**: Request the address of the 20 KB buffer (5 4 KB lines) for the dump file information record (DFIR). This record contains the general purpose registers, PSW, and dump ID storage at the time the dump was taken.

  These three fields will be at the offsets specified in the HCPDFIR format.

  Table 13. HCPDFIR Control Block

Field Description	Length	Offset
General Purpose Registers	64 bytes	X'0'
PSW	8 bytes	X'230'
Dump ID	100 bytes	X'240'

  Note: This subcode is intended for use by exit routines that have been coded to be used with the VM Dump Viewing Facility.

```
PLIST    DS       0F
         DC       AL4(*-*)    Address of the 20 KB buffer (5 4 KB lines) to be
                              used to return the HPCDFIR
         DC       H'51'       The code field
```

This record may not have any value for a virtual machine dump. The value of this record should be determined by the user. All fields should be referenced using their DSECT names.

Return codes for register 15 are:

Return Code
Explanation

0
Successful operation

4
Internal error

- **Code = 60**: Request PRTDUMP to print a buffer that has been translated.

```
PLIST     DS       0F
          DC       AL4(*-*)    The address of the buffer to be printed
          DC       H'60'       The code field
          DC       H'0'        The number of character lines
```

The buffer contains translated data with a fixed length of 133 characters, including prefixed print control code.

Return codes for register 15 are:

Return Code
Explanation

0
Successful operation

4
The number of lines = 0

500
Print failure

- **Code = 70**: Request PRTDUMP to print the registers and PSWs and the appended map.

```
PLIST     DS       0F
          DC       AL4(*-*)    Reserved
          DC       H'70'       The code field
```

Return codes for register 15 are:

Return Code
Explanation

0
Successful operation

100
Read error

500
Print failure

Note: When the map is not appended, register 15 equals zero and Dump Viewing Facility message 300I is issued.

- **Code = 71**: Request PRTDUMP to format and print the appended map.

```
PLIST     DS       0F
          DC       AL4(*-*)    Reserved
          DC       H'71'       The code field
```

Return codes for register 15 are:

Return Code
Explanation

0
  Successful operation

100
  Read error

500
  Print failure

Note: When the map is not appended, register 15 equals zero and Dump Viewing Facility message 300I is issued.

- **Code = 80**: Change the register set and PSW in the dump file information record in the dump.

```
PLIST     DS       0F
          DC       AL4(*-*)    The address of the buffer containing the register
                               set and the PSW
          DC       H'80'       The code field
```

Note: Used when the dump information record does not contain a valid virtual machine register set and PSW. The registers and PSW provided by the extraction are placed into the dump information record by the Dump Viewing Facility.

Return codes for register 15 are:

Return Code
  Explanation

0
  Successful operation

4
  Internal error

- **Code = 90**: Return to the user a module and an entry point name when given an address.

```
PLIST     DS       0F
          DC       AL4(*-*)    The address of the module or entry name
          DC       H'90'       The code field
          DC       H           Reserved
          DC       CL8''       The entry name returned to the caller
          DC       AL4(*-*)    The entry address to the caller
          DC       CL8''       The module name to the caller
          DC       AL4(*-*)    The module address to the caller
```

Return codes for register 15 are:

Return Code
  Explanation

0
  Successful operation

2
  Map not present

4
  Address not in map

100
  Read error

- **Code = 91**: Return to the caller an entry or module name address when given a name.

```
PLIST     DS       0F
          DC       AL4(*-*)    The address returned to the caller
          DC       H'91'       The code field
          DC       CL8'NAME'   The name field
```

The address is returned to caller.

Return codes for register 15 are:

Return Code	Explanation
0	Operation successful, address returned
2	Map not present
4	Name not found in map
100	Read error

PI end

Miscellaneous Services

PI

The Dump Viewing Facility provides the following non-SVC 199 services:

- DMMCLR (alias DTVCLR): Simulate clearing the screen

  The entry point provides the proper entry and exit linkage. The exit routine should branch to the entry point using a BALR instruction with registers loaded as follows:

Registers	Contents
0-12	Not specified
13	Save area address
14	Return address
15	Entry point address

  Description: Routine to clear screen upon call to DMMFEDLN.

  This entry point does not set a return code.

- DMMFEDLN (alias DTVFEDLN): Display data on screen.

  The entry point provides the proper entry and exit linkage. The exit routine should branch to the entry point using a BALR instruction with registers loaded as follows:

Registers	Contents
0	Not specified
1	A pointer to parameter list
2-12	Not specified
13	The save area address
14	The return address
15	The entry point address

The parameter list should be set up as follows:

```
PLIST     DC       AL4         The address of the data to be displayed
          DC       F           The number of bytes to be displayed
```

Description: Used to enable formatting routines to write to the screen during the DUMPSCAN session.

This entry point sets the following return codes:

Return Code
  Explanation

0
  The service performed

8
  PRINTL failure

- DMMHEX: Convert EBCDIC to hexadecimal.

  The entry point provides the proper entry and exit linkage. The exit routine should branch to the entry point using a BALR instruction with registers loaded as follows:

  Registers
    Contents

  0-3
    Not specified

  4
    The pointer to the input string

  5
    Reserve for output information (will contain the pointer to the output string)

  6-12
    Not specified

  13
    The save area address

  14
    The return address

  15
    The entry point address

  Description: Used to convert EBCDIC data to hexadecimal data and perform hexadecimal data validity checking.

  Note: The input string should be greater than 0, but less than or equal to 8 bytes in length.

  This entry point sets the following return codes:

  Return Code
    Explanation

  0
    Service performed

  8
    Invalid hexadecimal data

- DMMINT (alias DTVINT): Convert binary to EBCDIC

  The entry point provides the proper entry and exit linkage. The exit routine should branch the entry point using a BALR instruction with registers loaded as follows:

  Registers
    Contents

  0-2
    Not specified

3
The input byte count

4
The pointer to the input string

5
Reserve for output information (will contain pointer to output string)

6-12
Not specified

13
The save area address

14
The return address

15
The entry point address

Description: Used to convert binary data to EBCDIC data and perform byte count validity checking. If the byte count is greater than X'80', it will set it to X'80'; if less than 1, it will set it to 4.

This entry point will set the return code to zero, meaning that the service has been performed.

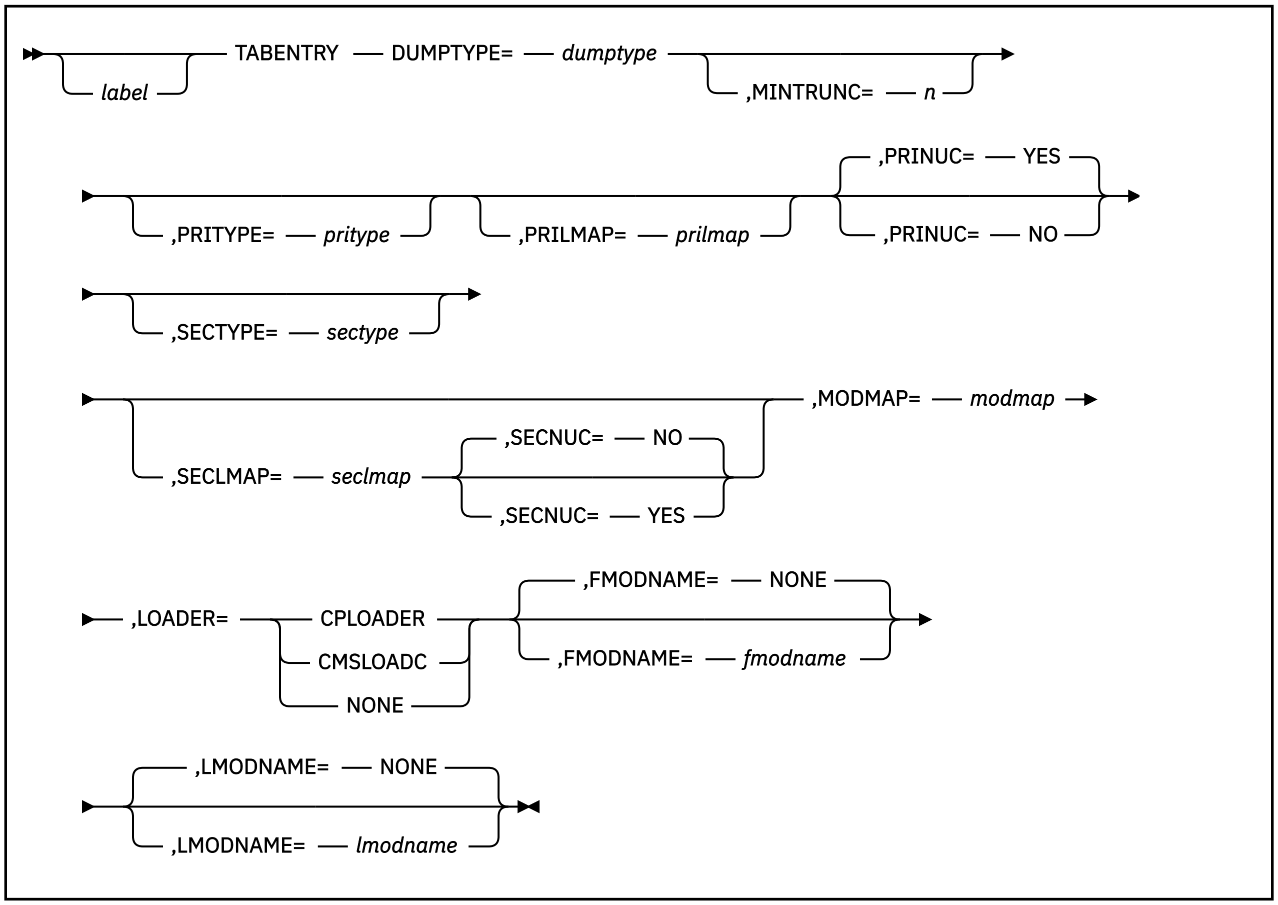
PI■end

Appendix C. Dump Viewing Facility Utilities

The following utilities are provided by the Dump Viewing Facility:

- TABENTRY Table Macro
- TBLENTRY Table Macro
- BLOCKDEF Utility
- BLOCKMAP Macro

TABENTRY Utility Macro



Purpose

Code the TABENTRY macro to generate a map information entry in the HCSTAB table for a specific dump type. This entry provides the information required for the compression of the dump's associated load map(s) that then can be used to generate a module map for the dump. The module map can then be appended to the dump by using the Dump Viewing Facility ADDMAP command.

Operands

DUMPTYPE=*dumptype*
: is a required 1- to 8-character string specifying the type of dump for which you want a module map created. There is no verification checking.

MINTRUNC=*n*
: is an optional number from 1 to 8 specifying the **min**umum **trunc**ation length of the character string specified by DUMPTYPE. This value cannot be greater than the length of the value specified by DUMPTYPE. If DUMPTYPE has no minimum truncation, you need not specify this, and a default of the length of the character string specified for DUMPTYPE is taken.

PRITYPE=*pritype*
: is an optional 1- to 8-character string specifying the **type** of load map that will be used as the **pri**mary load map. The default value is the same as the value specified for DUMPTYPE.

PRILMAP=*prilmap*
: is an optional 1- to 8-character string specifying the CMS file name of the input **pri**mary **l**oad **map** that is needed to create a module map for the specified DUMPTYPE. The default value is the same as the value specified for PRITYPE.

Note: Validity checking, such as whether this is a valid CMS file name or whether it is a complete and valid load map, is not done during the assembly of this macro. It is performed during Dump Viewing Facility MAP command processing.

PRINUC=YES
PRINUC=NO
is an optional operand specifying whether the **pri**mary load map is a **nuc**leus. If you do not specify PRINUC, the default is YES.

SECTYPE=*sectype*
is an optional 1- to 8-character string specifying the **type** of load map that will be used as the **sec**ondary load map. If there is no secondary load map, SECTYPE must not be specified. The default is NONE.

SECLMAP=*seclmap*
is an optional 1- to 8-character string specifying the CMS file name of the input **sec**ondary **l**oad **map** that may be needed to create a module map for the specified DUMPTYPE. If no secondary load map is needed (SECTYPE is not specified), SECLMAP must not be specified. If a secondary load map is needed (SECTYPE is specified), and SECLMAP is not specified, it will default to the value of SECTYPE.

Note: Validity checking, such as whether this is a valid CMS file name or whether it is a complete and valid load map, is not done during the assembly of this macro. It is performed during Dump Viewing Facility MAP command processing.

SECNUC=YES
SECNUC=NO
is an optional operand specifying whether the **sec**ondary load map is a **nuc**leus. If no secondary load map is needed (SECTYPE is not specified), SECNUC must not be specified. If a secondary load map is needed (SECTYPE is specified) and SECNUC is not specified, it defaults to NO.

MODMAP=*modmap*
is a required 1- to 8-character string specifying the CMS file name of the output module map created by the Dump Viewing Facility MAP command.

Note: Validity checking, such as whether this is a valid CMS file name or whether one does not already exist, is not done during the assembly of this macro. It is performed during Dump Viewing Facility MAP command processing.

LOADER=CPLOADER
LOADER=CMSLOADC
LOADER=NONE
is a required operand specifying how the load map for the DUMPTYPE was created. If the load map was created using either of the CP loaders (HCPLDR or DMKLD00E), specify CPLOADER. If the load map was created using the CMS Load command, specify CMSLOADC. If there was no load map created when loading (or installing) the software or one was generated in some other way, specify NONE.

FMODNAME=*fmodname*
FMODNAME=NONE
is an optional 1- to 8-character string specifying the first module's name in the load map for the DUMPTYPE specified. If there was no load map created when loading (or installing) the software or one was generated in some other way, specify NONE. The default is NONE; however, if LOADER was not specified as NONE, this field is required and must not have the value of NONE.

LMODNAME=*lmodname*
LMODNAME=NONE
is an optional 1- to 8-character string specifying the last module's name in the load map for the DUMPTYPE specified. If there was no load map created when loading (or installing) the software or one was generated in some other way, specify NONE. The default is NONE; however, if LOADER was not specified as NONE, this field is required and must not have the value of NONE.

Usage Notes

1. To code this macro, you should have a general knowledge of the HLASM Version 1 Release 3 Licensed Program. Specifically you should know what columns to use and how to continue a line. For more information, see *HLASM MVS & VM Language Reference V1 R3*.
2. The Dump Viewing Facility MAP command supports the compression of load maps created only by the following:
   - The HCPLDR loader
   - The DMKLD00E loader
   - The CMS Load command.

   All other load maps are not supported by the Dump Viewing Facility MAP command. To append an unsupported map to a dump, see Appendix D, "Module Map Architecture (Used by ADDMAP)," on page 189.
3. The first operands of this macro:
   - MINTRUNC
   - PRITYPE
   - PRILMAP
   - PRINUC
   - SECTYPE
   - SECLMAP
   - SECNUC
   - MODMAP

   are used in creating the MODULE MAP for the entry's DUMPTYPE.

   The last operands:
   - LOADER
   - FMODNAME
   - LMODNAME

   are used in compressing a LOAD MAP of the entry's DUMPTYPE.
4. The MINTRUNC operand, if specified, must be less than or equal to the length of the value specified for DUMPTYPE, and must be greater than zero.

   Note: Take care when establishing a minimum truncation for the DUMPTYPE to ensure that this abbreviation is still unique among the other DUMPTYPE entries.
5. If SECTYPE is not specified, SECLMAP and SECNUC must not be specified.
6. The PRINUC and SECNUC operands do not change the MAP command processing; instead they are used in the messages issued during MAP command processing.

   Following are examples of how the Dump Viewing Facility MAP command uses the PRINUC operand of this macro:

   a. If you have an entry with **DUMPTYPE=XYZ** and you specify **PRINUC=YES** (or do not specify it at all so that it defaults to YES), during MAP processing message HCS0121A prompts you as follows:

      ```
      ENTER THE FILENAME FILEMODE FILETYPE OF THE INPUT XYZ NUCLEUS
      LOAD MAP, A NULL LINE, SUBSET OR HX
      ```
   b. If you instead specify **PRINUC = NO**, during MAP processing message HCS0121A prompts you as follows:

      ```
      ENTER THE FILENAME FILEMODE FILETYPE OF THE INPUT XYZ LOAD MAP, A
      NULL LINE, SUBSET OR HX
      ```

   This scenario is similar for the SECNUC operand.

7. If a particular DUMPTYPE entry does not have a primary or a secondary load map of the same type and no other DUMPTYPE entry needs this entry's load map to create its module map, all of the following operands—LOADER, FMODNAME, and LMODNAME—should be specified as NONE.
8. If a particular DUMPTYPE entry has a primary or a secondary load map of the same type and another DUMPTYPE entry needs this entry's load map to create its module map, all of the following operands —LOADER, FMODNAME, and LMODNAME—must not be specified as NONE.
9. If a particular DUMPTYPE entry specifies LOADER as NONE, FMODNAME and LMODNAME must be specified as NONE.
10. If an entry has a primary or a secondary load map type (operands PRITYPE or SECTYPE) that is not the same as the entry's DUMPTYPE, there must be a separate entry in the table for the different DUMPTYPE. This different DUMPTYPE entry must also be **previously** defined in the table.
11. The load maps are inputs to the MAP command; they need to be contained in CMS files with the file name as specified in operands PRILMAP and SECLMAP. When the MAP command uses them, it expects the CMS file type to be MAP. These files can reside on any accessed disk.
12. The module map is the output of the MAP command. When created, it is contained in a CMS file with the file name as specified in operand MODMAP. The MAP command creates the CMS file type as MAP with file mode A.

Examples

The following TABENTRY macro creates an RSCSNET entry for the HCSTAB table:

```
TABENTRY DUMPTYPE=RSCSNET,                                         X
         MINTRUNC=4,                                               X
         PRINUC=NO,                                                X
         MODMAP=RSCSDVF,                                           X
         LOADER=CPLOADER,                                          X
         FMODNAME=DTMVEC,                                          X
         LMODNAME=DTMINI
```

The X's are used as continuation characters in column 72.

Note: The RSCSNET entry only requires a primary map to create its module map. The primary load map type and primary load map file name are the same as the DUMPTYPE, so PRITYPE and PRILMAP are not specified. Also the RSCSNET entry has a minimum truncation of 4. This allows MAP command processing to recognize all of the following as valid identifiers for RSCSNET:

- RSCS
- RSCSN
- RSCSNE
- RSCSNET.

The following TABENTRY macro creates a TSAF entry for the HCSTAB table:

```
TABENTRY DUMPTYPE=TSAF,                                            X
         PRITYPE=CMS,                                              X
         PRILMAP=CMSNUC,                                           X
         SECTYPE=TSAF,                                             X
         MODMAP=TSAFDVF,                                           X
         LOADER=CMSLOADC,                                          X
         FMODNAME=ATSCTL,                                          X
         LMODNAME=ATSVTT
```

The X's are used as continuation characters in column 72.

Note: The TSAF entry requires both a primary and a secondary load map to create its module map. Because the primary map is not of type TSAF but is of type CMS, a TABENTRY macro for DUMPTYPE = CMS must be defined before this one. Because the secondary load map CMS file name is the same as the secondary load map type specified for SECTYPE (TSAF), SECLMAP is not specified.

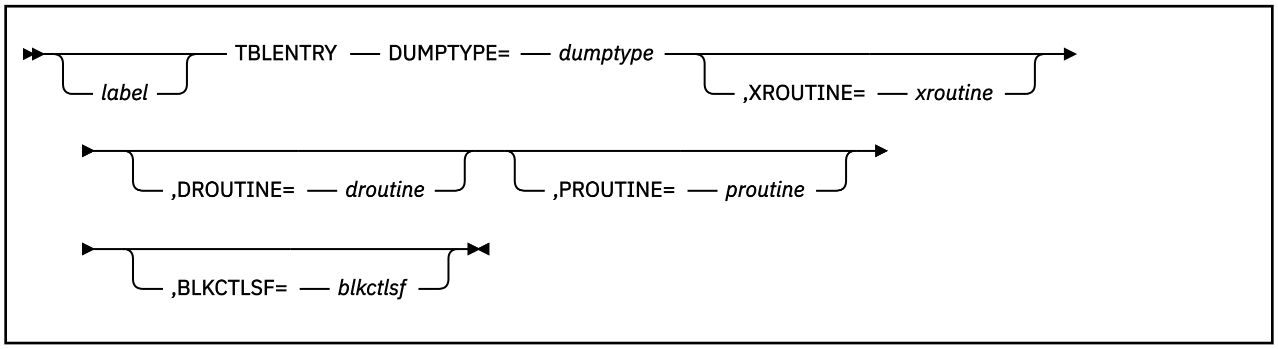
The following TABENTRY macro creates an RSCS entry for the HCSTAB table:

```
TABENTRY DUMPTYPE=RSCS,                                               X
              PRITYPE=GCS,                                            X
              PRILMFN=GCSNUC,                                         X
              MODMAP=GCSDVF,                                          X
              LOADER=NONE
```

The X's are used as continuation characters in column 72.

Note: The RSCS entry requires a primary load map of GCS; therefore, a TABENTRY macro for DUMPTYPE=GCS must be defined before this one. Because the primary map type is not RSCS and there is no secondary map required for RSCS (and also assuming no other entry requires an RSCS load map to produce its module map), the field LOADER is specified as NONE, and FMODNAME and LMODNAME are not specified (and thus defaulted to NONE).

TBLENTRY Utility Macro



Purpose

Code the TBLENTRY macro to generate an external file information entry in the HCSTBL table for a specific dump format. This entry provides the necessary extraction and formatting information for a specific dump type.

Operands

DUMPTYPE=*dumptype*
is a required 1- to 8-character string specifying the dump type for which this entry is used.

One of the following operands should be specified:

XROUTINE=*xroutine*
is an optional 1- to 8-character string specifying the CMS file name of the data extraction routine to be called from the Dump Viewing Facility for the specified DUMPTYPE.

DROUTINE=*droutine*
is an optional 1- to 8-character string specifying the CMS file name of the dump formatting routine that is called from the DUMPSCAN command for the specified DUMPTYPE.

PROUTINE=*proutine*
is an optional 1- to 8-character string specifying the CMS file name of the dump formatting routine that is called from the PRTDUMP command for the specified DUMPTYPE.

BLKCTLSF=*blkctlsf*
is an optional 1- to 4-character string specifying the BLOCK Control suffix of the file's file name for the specified DUMPTYPE. The file prefix of the BLOCK control file name is HCS$. The file type of the CMS BLOCK control file is TABLE. For example, if you specify DUMPTYPE=ABCDEFGH and BLKCTLNM=ABC, the file HCS$ABC TABLE is used as the block control table file for the entry ABCDEFGH. See "BLOCK Subcommand" on page 66 for additional information on the BLOCK subcommand control files.

Usage Notes

1. To code this macro, you should have a general knowledge of the HLASM Version 1 Release 3 Licensed Program. Specifically you should know what columns to use and how to continue a line. For more information, see *HLASM MVS & VM Language Reference V1 R3*.
2. Although every operand except DUMPTYPE is optional, you must have one operand other than DUMPTYPE to make the entry meaningful.
3. Validity checking such as whether operands are valid CMS file names and whether files are actually available on accessed disks are not performed during the assembly of this macro. This checking is performed during Dump Viewing Facility command processing.
4. For the XROUTINE, DROUTINE, and PROUTINE operands, you are specifying what the file name will be for the data extraction and dump formatting routines. The Dump Viewing Facility commands expect the file type of these routines to be TEXT. These files can exist on any accessed disk.

5. For the BLKCTLSF operand, you are specifying what the suffix of the CMS file name is for the BLOCK control file. The Dump Viewing Facility Block subcommand of the DUMPSCAN command expects the file name prefix of this file to be HCS$ and the file type to be TABLE. This file can exist on any accessed disk.

Examples

The following macro creates a GCS/XA entry in HCSTBL with all the fields specified.

```
TBLENTRY DUMPTYPE=GCS,                                         X
         XROUTINE=GCTIEX,                                      X
         DROUTINE=GCTIDS,                                      X
         PROUTINE=GCTIPR,                                      X
         BLKCTLNM=GCS
```

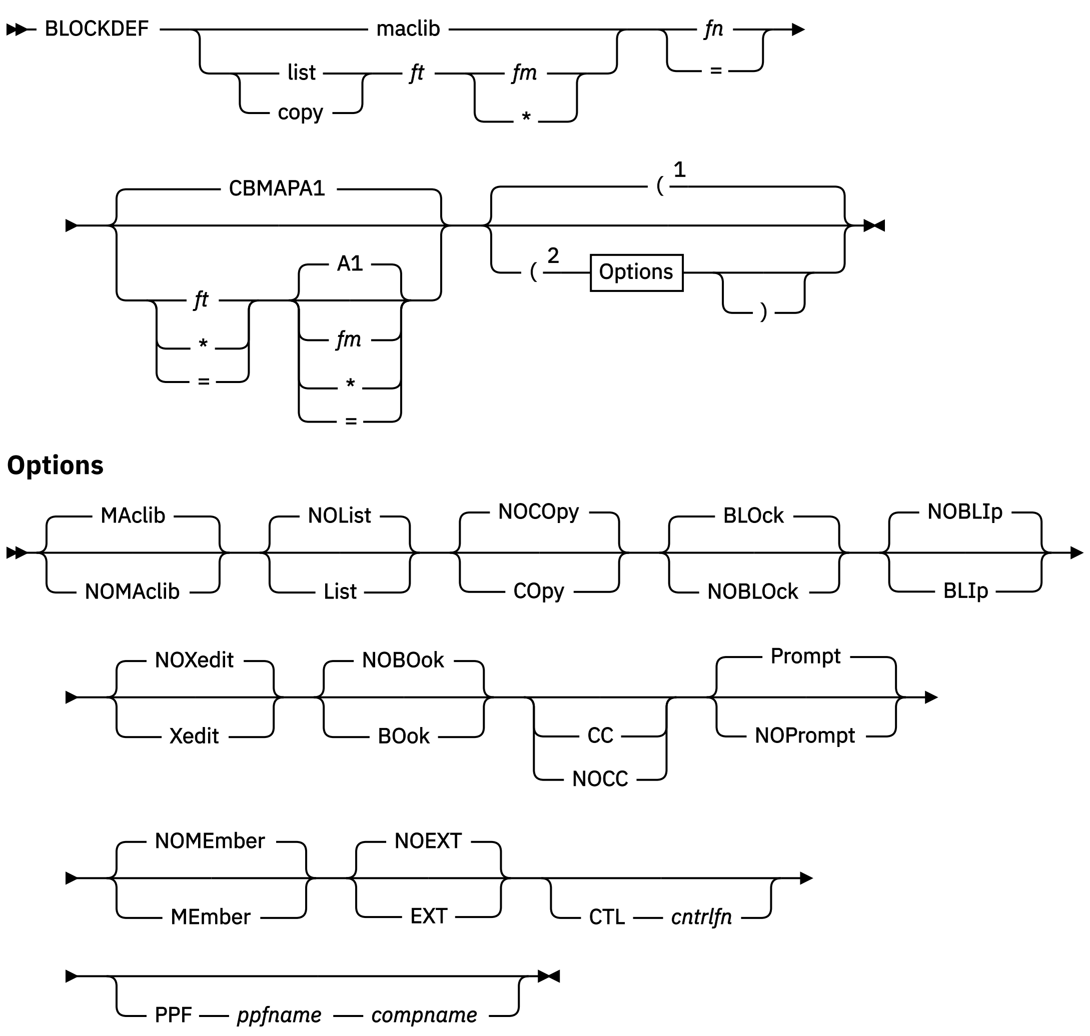
The X's are used as continuation characters in column 72.

The following macro creates an ABC entry in HCSTBL with only its format routine from the DUMPSCAN subcommand specified. All other fields are set to their respective defaults.

```
TBLENTRY DUMPTYPE=ABC,                                         X
         DROUTINE=ABSXYZ
```

The X's are used as continuation characters in column 72.

BLOCKDEF Utility Command



Notes:

[1] The defaults you receive appear above the line in the Options fragment.

[2] You can enter Options in any order between the parentheses.

Purpose

Use the BLOCKDEF utility command to generate dummy section (DSECT) format files for the DUMPSCAN BLOCK subcommand and to generate DSECT information print files for users.

Note: The output from the BLOCKDEF utility command is z/VM product implementation information for diagnosis and must not be used for programming purposes.

The BLOCK subcommand uses the format file to format data from a z/VM dump. See "BLOCK Subcommand" on page 66 for the description and format of the DUMPSCAN BLOCK subcommand.

Operands

maclib
is the file name of a MACLIB of control blocks to be processed by BLOCKDEF. The file type and file mode will default to MACLIB and *, respectively.

list
: is the file identifier (file name, file type, and file mode) of a file list of control blocks to be processed by BLOCKDEF. The format is similar to CMS EXEC but without the EXEC variables *&1 &2*.

copy
: is the file identifier (file name, file type, and file mode) of a file containing a single control block.

fn
: is the file name for the generated book/block file. If you specify =, the file name of the input file is used.

ft
: is the file type for the generated book/block file(s). If you specify * or nothing, the default CBMAP (for block files) or LISTING (for print files) is used. CBMAP and LISTING are the file types of the book/block file(s) shipped with z/VM If you specify =, the file type of the input file is used.

fm
: is the file mode for the generated book/block file(s). If it is not specified, A1 is used. If the file mode is not accessed as a read/write disk, processing ends.

Options

MAclib
: tells BLOCKDEF that the input file name is the name of a MACLIB containing the control blocks to be processed. This is the default.

NOMAclib
: suppresses the MACLIB option. If you use the NOMACLIB option, you must also use the COPY or LIST option.

NOList
: suppresses the LIST option. This is the default.

List
: tells BLOCKDEF that the input file name is the name of a file containing a list of the control blocks to be processed. The list can be in CMS EXEC file format, meaning that the first five characters on each line are *&1 &2*, the same format as found in a CMS EXEC file (See *z/VM: CMS Commands and Utilities Reference*, “LISTFILE Command”, for details). The list must consist of individual CMS data sets, not MACLIB members. To process specific MACLIB members, see the MEMBER option.

NOCOpy
: suppresses the COPY option. This is the default.

COpy
: tells BLOCKDEF that the input file name is the name of a single control block to be processed.

BLOck
: tells BLOCKDEF to build the control block file used by the BLOCK subcommand of DUMPSCAN (a Dump Viewing Facility command). If NOBLOCK and NOBOOK are used together, all output is canceled. This is the default.

NOBLOck
: suppresses the BLOCK option. If NOBLOCK and NOBOOK are used together, all output is cancelled.

NOBLIp
: suppresses the BLIP option. This is the default.

BLIp
: tells BLOCKDEF to display, on the user terminal, the name of the control block about to be processed.

NOXedit
: suppresses the XEDIT option. This is the default.

Xedit
: tells BLOCKDEF to display the generated output in an XEDIT session, rather than in a CMS file. If you wish to save this information, enter an XEDIT FILE command. If you specify XEDIT with the MACLIB option (the default input file type), each member of the MACLIB is displayed after it is processed.

NOBOok
suppresses the BOOK option.

BOok
tells BLOCKDEF to generate a CMS file that is formatted for printing. Each control block is formatted with its prolog and all block comments. A storage layout diagram and control block field index are added to the default BLOCK format. This format is not intended for use by the BLOCK subcommand.

CC
tells BLOCKDEF to place basic carriage control characters into the file to be printed. The CC option is valid only when you also specify the BOOK option. It is ignored at all other times. When the BOOK option is used, CC becomes the default.

NOCC
suppresses the CC option. When the BOOK option is **not** used, NOCC is the default.

Prompt
tells BLOCKDEF to issue a prompt for permission to continue processing with the files indicated as the input and output file. This is the default.

NOPrompt
suppresses the PROMPT option.

NOMEmber
suppresses the MEMBER option. This is the default.

MEmber
tells BLOCKDEF that you want to be prompted for the names of control blocks to be extracted from the MACLIB and processed. This option is valid only when the MACLIB option is specified. It is ignored at all other times. It is not affected by the PROMPT or NOPROMPT option.

NOEXt
suppresses the EXT option and specifies that the $TBKDUXT user exit code should not be used to locate external copy files. This is the default.

EXT
tells BLOCKDEF that external copy files should be located only with the user supplied exit code in $TBKDUXT EXEC. See usage note "18" on page 174 for further details on the interface between BLOCKDEF and the user exit code.

CTL *cntrlfn*
specifies a control file name to be used by BLOCKDEF to resolve external copy files. BLOCKDEF will search for external copy files in each maclib on the MACS statement in the control file.

cntrlfn
is the name of the control file used to identify the MACLIB list. The file type of the control file is CNTRL.

PPF *ppfname compname*
specifies a SES product parameter file name and component to be used by BLOCKDEF to resolve external copy files. BLOCKDEF will use the product parameter file to locate the component's control file. The MACS statement in the control file will then be used to find the list of maclibs to be searched for external copy files.

ppfname
is the file name of the product parameter file. Its file type must be PPF.

compname
is the name of the component associated with the *ppfname*, such as CP or CMS.

Usage Notes

1. BLOCKDEF is intended to process dummy sections (DSECTs) written for the IBM System/370 assembler.
2. The BLOCKDEF options are processed from left to right. If you specify an option twice, the rightmost occurrence of the option is used.

3. When the *list* or *copy* file name is specified, the file type and file mode must also be specified. The file mode can be specified as an * to search all accessed disks for files with the specified file name and file type.
4. The BOOK and BLOCK options are mutually exclusive. If you specify these options together, an informational message is issued and the rightmost option prevails.
5. When the book option is used with this utility, BLOCKMAP is invoked and all BLOCKMAP conventions apply.
6. If you are using XEDIT to edit a control block and want to run BLOCKDEF against that file, it is not necessary to provide the input and output file names. The input file defaults to the file you are editing, and the output file defaults to the input file name and a file type of CBMAP.
7. The amount of virtual storage you need for this command depends on the largest control block being processed.
8. Book/block files can be very large. The size of the files depends on the commenting style used in the control blocks and the options specified. You should have a write access disk with enough space to contain it.
9. Control blocks that are written as macros are ignored.
10. BLOCKDEF erases the old output files (if they exist) before processing begins. If you wish to keep the old book/block files, move them to another disk or rename them.
11. The control blocks being processed do not have to be on a write access disk.
12. If a DSECT operation code is not found, the label on the first DC or DS operation code is used as the control block name.
13. EQU statements that use calculations for operands are ignored. Symbols that are equated to another symbol, which in turn is equated to yet another symbol, are also ignored.
14. The file BLOCKDEF RUNLOG is updated whenever BLOCKDEF is issued. It contains the names of all control blocks processed, along with any informational or error messages issued while the control block is being processed.
15. The *list* option input file must have at least two names on each line. The first name is used as the file name for the input copy file. The second name is used as the file type of the input copy file. If a third name is found on the line, it is used as a file mode. Comments in the input file are indicated with an asterisk (*) in column 1.
16. Invalid options are indicated by an informational response. Processing proceeds unless you respond with a NO when prompted.
17. EXTERNALLY REFERENCED COPY FILES

    BLOCKDEF attempts to resolve references to external copy files as described in the 'Bit and Code Definition Tables' section of the BLOCKMAP command description. The external copy files are located by one or more of the following methods:

    - by a user supplied exit routine called $TBKDUXT EXEC
    - as a member of the maclib specified on the BLOCKDEF command
    - as a member of one of the maclibs in the MACS statement of a control file. These methods are controlled by the MACLIB, BOOK, EXT, CTL, and PPF command options.
18. The $TBKDUXT USER EXIT

    $TBKDUXT is a user supplied exec for locating external copy files. BLOCKDEF invokes $TBKDUXT in one of two situations:

    a. the EXT option was specified, meaning that $TBKDUXT should be the exclusive method for locating external copy files.
    b. all other methods for locating the external reference have failed and NOEXT was not specified.

    BLOCKDEF passes to $TBKDUXT the file name of the external copy file to be located. $TBKDUXT must locate the copy file, place it in the XEDIT ring, and change its file type to 'DEFINED'. If $TBKDUXT returns to BLOCKDEF without locating the copy file, BLOCKDEF will end if the BOOK option was specified. Otherwise, the external reference will be ignored and BLOCKDEF will continue processing. $TBKDUXT should return code 0 if the external copy file was found. Any non-zero return code indicates that $TBKDUXT was unable to find the external copy file.

As an example, the following $TBKDUXT EXEC could be used to check the user's A-disk for external copy files:

```
/* $TBKDUXT EXEC to check my A-disk for external copy files when  */
/* running BLOCKDEF                                                */
parse upper arg fn.
/* Preserve the user's settings for CMSTYPE                        */
If cmsflag('CMSTYPE') = 1 Then user_cmstype = 'RT'
else user_cmstype = 'HT'
/* Check to see if the copy file is on the A-disk.                 */
address 'CMS' 'SET CMSTYPE HT'
address 'CMS' 'STATE fn 'COPY A'
if rc = 0 then do
  Address 'XEDIT' 'XEDIT' fn 'COPY (NOPROF'
  Address 'XEDIT' SET FTYPE DEFINED'
  Address 'CMS' 'SET CMSTYPE' user_cmstype
  end
/* Not found so exit to caller with error return code              */
address 'CMS' 'SET CMSTYPE' user_cmstype
Say '$TBKDUXT was unable to find fn 'COPY.'
Exit 12
/* end of $TBKDUXT EXEC example                                    */
```

19. RESOLVING EXTERNAL REFERENCES

    The EXT option causes BLOCKDEF to call the user exit code in $TBKDUXT EXEC as the only resolution method. If the user exit code fails to locate the external reference, BLOCKDEF's response will depend on the BOOK option. If BOOK was specified, then BLOCKDEF will end processing. Otherwise BLOCKDEF will ignore the external reference and continue processing with no output representing the reference to the external copy file.

    If the EXT option is not specified, and BLOCKDEF is processing with the MACLIB or MEMBER options, the first resolution method will be to search for the external copy file within the maclib specified on the BLOCKDEF command line. If the external copy file is not located in the maclib, the next search method will depend on the CTL or PPF options.

    The CTL option is used to specify a control file. BLOCKDEF will use the list of maclibs in the MACS statement from the control file to resolve external references. Each maclib in the list will be searched until the copy file is located or the list is exhausted.

    The PPF option is used to specify a SES product parameter file and a component name. These are used to locate the control file for the specified component. BLOCKDEF will use the list of maclibs in the MACS statement from the control file to resolve external references. Each maclib in the list will be searched until the copy file is located or the list is exhausted.

    The CTL, PPF and EXT options are mutually exclusive because the external resolution methods specified by each may conflict. If more than one of these options is specified, BLOCKDEF will issue an informational message and use the rightmost option from the command. If none of the previous methods listed have been able to locate the external copy file, and NOEXT was not specified, the $TBKDUXT EXEC will be invoked, if it exists.

    If all methods fail to locate the external copy file, the action of BLOCKDEF depends on the BOOK option. If the BOOK option was specified, BLOCKDEF will end processing. If the BOOK option was not specified, BLOCKDEF will ignore the external reference and continue processing with no output representing the reference to the external copy file.

Responses

- *** Extra characters “*nnnnnnnn*” ignored

  Explanation: This response shows that the indicated operands are ignored. Any other operands and options are accepted.

  User response: No action is required.

- *** Options *optionL* and *optionR* are mutually exclusive

  Explanation: The options named are mutually exclusive. *optionR* was the rightmost option and prevails.
- *** These files have the input filename and filetype specified

  Explanation: This response indicates that there is more than one file with the file name and file type you specified. BLOCKDEF uses the topmost file.
- *** Using "*fn ft fm*" as the input file

  Explanation: This response indicates the file to be processed.

  User response: No action is required.
- *** Using "*<fn ft fm>*" for the output block file

  Explanation: This response indicates the output block file that will be used.

  User response: No action is required.
- *** Using "*fn ft fm*" for the output book file

  Explanation: This response indicates the output book file that will be used.

  User response: No action is required.
- *** The file "*fn ft fm*" exists and will be erased

  Explanation: This response indicates that the output files you have specified already exist on the disk. BLOCKDEF erases these files before processing the control block(s).
- Do you want to proceed with the files shown? (Yes/No)

  Explanation: This prompt appears when an input file is found and the file mode of the output file specifies a read/write disk. This prompt lets you end processing if the input file identified by BLOCKDEF is not the one you want processed, or if the output file would overwrite one that you do not want changed.

  User response: Entering YES allows BLOCKDEF to continue. Entering NO ends BLOCKDEF processing.
- *** No maclibs were located to resolve external references

  Explanation: The command options included EXT, CTL or MACLIB, suggesting that you expect that one or more maclibs are to be used in resolving external references, but no maclibs have been located.

  User response: Verify that the arguments to the EXT or CTL options were specified correctly.
- *** The following maclibs will be used to resolve external references

  Explanation: The command options included EXT, CTL or MACLIB, and this message lists the maclibs which will be used to resolve external references.

  User response: No action is required.

Messages and Return Codes

- *** An output file identifier must be specified

  Explanation: This message indicates that an output file name is required.

  User response: Try the command again.

  Severity: 8
- *** An output file identifier may not be *

  Explanation: This message indicates that an output file name may not be specified as "*".

  User response: Try the command again.

  Severity: 8
- *** An input file type is needed with the LIST or COPY options

Explanation: This message indicates that an input file type is required when using the LIST or COPY options.

User response: Try the command again.

Severity: 8

- *** An input file identifier may not be =

  Explanation: This message indicates that an input file name may not be specified as "=".

  User response: Try the command again.

  Severity: 8

- *** Blockdef is not supported for CP control blocks
  *** Look for similar functions in VMDUMPTL

  Explanation: This message indicates a standard CP maclib name was specified as the input file name. BLOCKDEF can no longer be used with CP control blocks.

  User response: Use the VM Dump Tool when using CP control blocks.

  Severity: 50

- *** The disk "*nn*" is not a write access disk

  Explanation: This message indicates that the disk you have chosen for the output files appears to be read-only.

  User response: Specify another output file mode and try the command again.

  Severity: 38

- *** The input file *fn ft* could not be found on any accessed disk

  Explanation: This message indicates that the input file you specified could not be found.

  User response: Check to see that the file exists (and that you have spelled it correctly) and try the command again.

  Severity: 8

- *** Return code *nnn* from STATE command. *** Invalid file identifier "*fn ft fm*"

  Explanation: This message indicates that processing cannot continue because the CMS STATE command found the file identifier objectionable. (See *z/VM: Other Components Messages and Codes* for an explanation of the return code.)

  User response: Try the command with a different file identifier.

  Severity: *nnn* (from STATE command)

- *** Error *nnn*, from EXECLOAD command

  Explanation: This message indicates that processing cannot continue because the CMS EXECLOAD command failed while loading a macro. (See *z/VM: Other Components Messages and Codes* for an explanation of the return code.)

  User response: Try the command with a different file identifier.

  Severity: *nnn* (from EXECLOAD command)

- *** An input file must be specified

  Explanation: This message means that an input file name file type is required.

  User response: Try the command again.

  Severity: 8

- *** The MACLIB "*maclib*" member "*member*" was not found

  Explanation: This message means that the named MACLIB member could not be found in the named MACLIB.

User response: Try the command again with another MACLIB or correct your spelling of the member name.

Severity: 8

- *** NOBOOK and NOBLOCK options used. No output is possible.

  Explanation: This message means that the options used have suppressed all output.

  User response: Use the BOOK option to get a book format of the control block. Use the BLOCK option to get a format used by the BLOCK subcommand.

  Severity: 8

- *** COPY or LIST options must be used with the NOMACLIB option.

  Explanation: The contents of the input file are indeterminate when the NOMACLIB option is used without specifying the alternative COPY or LIST file types.

  User response: Retry the command with either the COPY or the LIST option.

  Severity: 8

- *** EXT option requires access to $TBKDUXT EXEC

  Explanation: The EXT option specified to BLOCKDEF indicates that externally referenced copy files should be resolved using the user supplied $TBKDUXT EXEC which was not found on any accessed disk.

  User response: Access the disk containing your $TBKDUXT EXEC. You may create your own $TBKDUXT EXEC according to the explanation provided in the Usage Notes section.

  Severity: 8

- *** No control file specified on CTL option

  Explanation: The CTL option specified to BLOCKDEF indicates that externally referenced copy files should be resolved using a control file, but none was specified.

  User response: Try the command again and specify a control file name immediately following the CTL option.

  Severity: 8

- *** PPF option requires product parameter file and component

  Explanation: The PPF option specified to BLOCKDEF indicates that externally referenced copy files should be resolved using a control file located from a product parameter file and component name. Both must be specified with the PPF option.

  User response: Try the command again and specify a product parameter file and component name immediately following the PPF option.

  Severity: 8

- *** Error finding control file name with VMFSIM

  Explanation: BLOCKDEF was unable to locate a control file using VMFSIM.

  User response: Refer to the VMFSIM documentation in the *z/VM: VMSES/E Introduction and Reference* to interpret any messages issued from VMFSIM.

  Severity: 8

- *** Control file *cntrlfn* CTL *fm* not found

  Explanation: The control file specified to BLOCKDEF using the CTL or PPF options could not be located.

  User response: Access the disk containing the control file and retry the command.

  Severity: *nnn*(from ESTATE on control file)

- *** Error reading control file *cntrlfn ft fm*

  Explanation: The control file specified to BLOCKDEF using the CTL or PPF options could not be read.

User response: Check the return code from EXECIO, correct the error, and retry the command.

Severity: *nnn*(from EXECIO on control file)

BLOCKMAP Macro

BLOCKMAP is an XEDIT macro used to map the data areas within DSECTs. These DSECTS can be either IBM supplied or user created so long as they are in ASSEMBLER source format. This macro accepts the data definitions and control statements as input and creates a formatted control block. This formatted control block includes the original input, a map of the data areas defined in the DSECT, and a map of the flags or code-defining EQUates. The formatted control block is a pictorial representation of the DSECT data areas formed by a series of comments, statements, and the original DSECT definition.

Note: The output from the BLOCKMAP macro is z/VM product implementation information for diagnosis and must not be used for programming purposes.

A major portion of any program's internal structure is used for creating and altering control blocks, and for passing information in the form of control blocks or data structures. An important element in the documentation is the support of this function. BLOCKMAP automates this documentation support by

- Accepting, as input, the data definition and control statements that define a control block
- Producing a map of all flag or code defining equates (EQUs) included in the control block, or of another control block referenced within the current one.

Note: BLOCKMAP and BLOCKDEF no longer support CP control blocks.

BLOCKMAP Conventions

The BLOCKMAP input conventions ensure that control blocks are readable and consistent. A control block is written in System/370 Assembler Language. Because BLOCKMAP creates a map of the control block, only one DSECT is allowed per copy file. A copy file is the file in which the control block is defined and is the term that will be used throughout this guide. The effectiveness of BLOCKMAP depends on adherence to these conventions. A description of these conventions is provided in the following sections.

PI

Field Naming: For notational convenience, control block field names have prefixes specified by 1 to 5 characters. The field names have the same prefix as the prefix on the DSECT label. BLOCKMAP will attempt to use the full name when drawing the control block map. For 1-byte fields, the maximum length is 6 characters. If the name is longer than this, BLOCKMAP will replace the prefix characters with a colon (:). If the resulting name is still too long for the field, BLOCKMAP will use the hexadecimal offset in parentheses as the name. The hexadecimal offset is the offset of the field from the beginning of the control block. In the case of field overflow (a field definition that occupies two lines of the control block image), the field name will appear in both the leading field as well as in the continuation field. The leading field name will have a dash (–) appended to it, while the continuation field name will be prefixed by a dash. If either the leading or continuation fields are 1 byte in length, the total length, including the dash, cannot exceed the 6 characters allowed for the field.

Definition of Reserved Fields: Reserved fields are denoted by a data operand (DS/DC) with no associated field name. Reserved fields are denoted in the control block pictorial as a group of slashes (/), while defined fields contain the field name in the pictorial. Two consecutive undefined fields will be depicted as two undefined fields in the control block pictorial and will not be logically combined to form a single field.

Control Block Structure/ORG Processing: The entire main body of the control block must be defined first. This main definition may have one or more ORG statements that may provide for additional detail for a larger field in the main picture or a complete redefinition of a field based on some device specification. Resetting the location counter to its original value, when the ORG statement is specified without operands, is not supported.

When an ORG statement is encountered, main picture processing is terminated. The data definition statements processed to that point will be appended after the commented statements forming the control block picture. The complete commented control block picture is the main body definition. BLOCKMAP will then enter **redefinition mode**. Each ORG followed by its data definition(s) will create a new picture for those fields. Redefinitions must not exceed the bounds of the main body definition. The target of an ORG statement must be a previously defined field and cannot be the label on the DSECT statement, another ORG statement, or an EQU statement.

As part of the redefinition picture, the following title line is built:

```
****     REDEFINITION -  .........
```

Any comments that appear on the ORG statement up to column 63 will be appended to the title line.

Redefinition Mode and Unnamed Fields: The treatment of unnamed reserved fields is handled differently in redefinition mode. At times, an unnamed field is encountered and the operands describing that field are syntactically equivalent to those appearing at the same displacement in a previous definition. In this case, the picture field generated will contain the name of the previous definition field and will not be depicted as being reserved. REACH-BACK will attempt to define an unnamed field based on the target field of the ORG statement. If this attempt fails, REACH-BACK uses the field name in the main body definition as the field name in the unnamed redefined field.

Variable Length Fields: A variable length field must be the last item in the definition. It is denoted by a replication factor of zero and the control phrase `START OF VARIABLE LENGTH DATA` in the comment area. The field must be named. Redefinition blocks can have a variable length field only if one was specified in the main definition. The field is drawn in the picture with a series of colons, and no ending address is given.

Operators Supported: BLOCKMAP supports the following operators. Unrecognized statements are passed through, but ignored.

DSECT
Only one accepted per copy file.

DS, DC
Define a block in the picture.

EJECT
See Picture Segmentation Option.

ORG
See Control Block Structure/ORG Processing.

CCW, CCW0, CCW1
Define a doubleword block in the picture.

Data Type Operands Supported: BLOCKMAP is designed to support only a subset of the possible data operand coding conventions. Specifically, only the following data type operands are accepted:

D
Doubleword

F
Fullword

H
Halfword

X
Hexadecimal byte

A
Address constant

V
External address constant

C
Character byte

The alignment implicit to the above operands will be enforced. This means that these operands must be on a boundary divisible by its implicit data length. An exception to this requirement is the presence of a length modifier that negates implicit length alignment. Automatic alignment, and, therefore, undefined holes within the control block, will not be performed. Replication factors on any of the above data types will be accepted. A replication factor of zero will cause the field name to be ignored by BLOCKMAP in the control block picture. However, the operand will be checked for proper alignment. Explicit length modifiers are accepted on data type operands A, C, and X. A replication factor greater than 1 is not accepted in combination with a length modifier for the A type operand; it is permissible for the C and X type operands. C and X type operands that have both a replication factor and a length modifier will be formatted as one contiguous entity. Its length will be equal to the product of the replication factor and length modifier. For example, 2XL2 will be pictorially represented as XL4. Only one data definition operand per statement will be processed. For example, A(0),A(0) will be interpreted as if only one address constant were specified.

Bit and Code Definition Tables: Bit definition tables can be built for all flag byte definitions in the control block, and code definition tables for value equates. The actual bit or code definitions can be contained in this control block, or defined in an external copy file, either formatted or not. When the bit or code equates are in the current file, table formatting is invoked by one of the following phrases in a comment line:

```
BITS  DEFINED IN fieldname
CODES DEFINED IN fieldname
```

When the bit or code equates are in an external file, the control phrases have the following format:

```
BITS DEFINED FOR local-fieldname BY file name fieldname
CODES DEFINED FOR local-fieldname BY file name fieldname
```

In the external file the bits or codes may be defined by the local form or another format. The local form implies that the actual space for the byte is contained in the structure. The other format is used for global definition copy files that do not themselves define any space.

```
* fieldname BIT DEFINITION
* fieldname CODE DEFINITION
```

Bit-defining equates have the following format and can be intermixed within the byte definition:

```
bitname  EQU    X'xx'                  description
bitname  EQU    bitname1+bitname2...   description
```

Code-defining equates have one of the following formats; decimal and hex codes cannot be intermixed, nor can 1- and 2-byte fields.

```
codename  EQU    X'xx'      description
codename  EQU    X'xxxx'    description
codename  EQU    n          description
```

where *x* is a hexadecimal number and *n* is a decimal number.

The first such statement found that fits the above model starts the definition table. Table processing is terminated by one of the following statements:

- A statement other than a comment, that does not match the format for the table; for example, a DS, ORG, or inappropriate EQU
- A comment statement containing another definition control phrase
- The comment statement `* END OF DEFINITION`
- End-of-file.

Note: If codes are defined in an external file, the last line of the external code file must be a comment. If codes/bits are defined in an external file, a comment line must follow the last field that is defined. BLOCKMAP needs this comment starting in column 1 to execute properly.

The field being described must have been defined before the first EQU statement for it. The field name may have any of the following special characters appended to it; they will be ignored.

- . (period)
- , (comma)

- - (hyphen)
- = (equal sign)
- : (colon)
- ; (semicolon)

The hexadecimal displacement of the field in the control block is noted in the title line or lines, which replace the invoking comment

```
****    BITS DEFINED IN fieldname (AT HEX DISPLACEMENT: addr)
****    CODES DEFINED IN fieldname (AT HEX DISPLACEMENT: addr)
```

or, for an externally defined set of values:

```
****    BITS DEFINED FOR local-fieldname BY file name fieldname
****    (AT HEX DISPLACEMENT: addr)
****    CODES DEFINED FOR local-fieldname BY file name fieldname
****    (AT HEX DISPLACEMENT: addr)
```

This title card will replace the `BITS DEFINED` statement encountered in the input stream. This implies that any additional characters following the field name on the input statement will be ignored. For those flags that are defined against more than 1 byte, multiple `DEFINED IN/FOR` statements should be used (one for each byte for which the definition applies). These keyword statements must appear before the first EQU model statement but may be separated from each other by intervening comment statements.

As shown in the model of bit-defining EQUs above, a bit name can be defined to have a value equal to the summation of some number of previously defined bit names. The scope of reference for these previously defined bit names is the current flag byte definition only. Bit names defined for other flag bytes cannot be referenced outside of the scope of their flag byte definition. Code definition EQU statements cannot reference labels.

The description portion of the statement can be continued to subsequent statements either by use of the continuation column or by the use of subsequent comment statements. All comment statements that follow an EQU model statement (except for one that contains a control phrase) will be treated as a description continuation.

PI end

BLOCKMAP Invocation

BLOCKMAP is an XEDIT macro that is invoked while editing a file containing the control block to be mapped. BLOCKMAP can be issued anywhere in the file, regardless of the current line orientation. Also, any number of files can be active in the XEDIT when BLOCKMAP is invoked.

In order for BLOCKMAP to run, some additional control statements are required. These control statements are found as special tags in the control block prolog. BLOCKMAP looks for a one-line description of the control block following the prolog tag `DESCRIPTIVE NAME:`. The remaining line is assumed to be the text to be used while building the control block structure. If this tag is not found, a blank line appears in the output. BLOCKMAP looks for the prefix length tag as the switch to indicate that the file should be mapped. The tag must appear as `PREFIX_LEN =`*n*, where *n* is the number of characters that prefix every field in the control block. This prefix is stripped off field names that may be just too long to fit in their storage declaration picture.

BLOCKMAP has many formatting capabilities. For brevity, not all examples are illustrated. A discussion of the formatting capabilities will precede each example of formatting.

Processing: Processing of the input file starts when the first DSECT statement is found. The control block name is the field name of the DSECT. Only one DSECT record per input file is accepted; all others will be ignored with the data following them being considered to be part of the original DSECT definition.

Upon successful completion of BLOCKMAP processing, the screen will be left in edit mode for the work file. It is your responsibility to save or file this work file if it is to be retained. If an error occurs during BLOCKMAP processing, an appropriate error message will be issued. The input file will be oriented such that the current line will be the line that caused the error. The work file created, if any, will be left intact, although it may contain incorrect images because of incomplete processing. There is no need to erase or quit this file before reinvoking BLOCKMAP.

Various Field Pictures: Control block fields are depicted in boxes with a length of 1 doubleword. When required, the doubleword field is segmented to form fullwords, half words, or bytes. The control block picture has the hexadecimal displacement of the doubleword boundary for any box space that starts a new field definition. All control blocks will also have their ending address printed. This address will be printed in the remaining space of an unused doubleword definition or on the line immediately following the last definition.

Fields that cross a doubleword boundary are represented as **spanned** records. A box space is continued on the next line by using hyphens (**-**) as continuation characters. See Figure 60 on page 184 for a picture. Large fields that use spanned records and share a common boundary will have that boundary removed.

The DS or DC field name is centered within the box it defines. A reserved field box will be filled with slashes (/). The field name of a spanned record will appear in both boxes. A field larger than two doublewords will use the standard doubleword box with break symbols (=) instead of vertical lines (|) for the box sides to represent more than one doubleword. This eliminates a huge blank box for fields many doublewords long. The field name of a field larger than a doubleword that shares a common boundary and does not have a break character sequence will be centered in the largest field of the definition.

An abbreviated field name will be used when the name of a byte field is greater than 6 characters. A colon (:) will be substituted for the prefix. If the abbreviated field name of a byte field is larger than 6 characters, the hexadecimal displacement will be placed in the field.

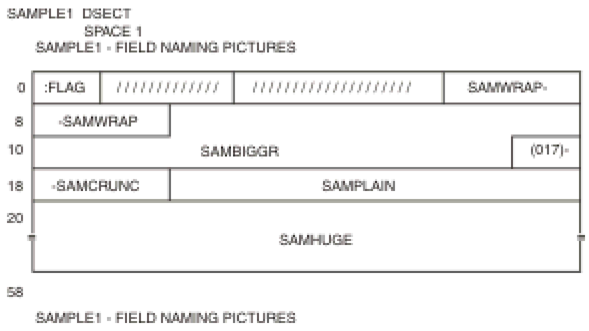


Figure 60. SAMPLE1 COPY Control Block

Redefinition Pictures: Any redefinition of fields must occur after the entire control block has been defined. When an ORG is encountered, main picture processing ends. Each ORG followed by its data definition will cause a new picture to be drawn. A title line,

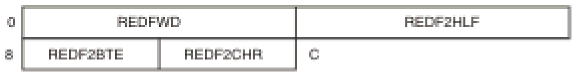
```
****     REDEFINITION -  .........
```

will be written before the picture is drawn.

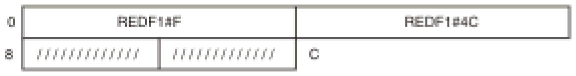
The starting displacement for redefinitions is defined by the ORG operand. For redefinitions that do not start on doubleword boundaries, the control block picture will be indented the appropriate number of spaces needed to represent the degree of offset from a doubleword boundary. The left-hand edge will contain the doubleword boundary address upon which the control block definition falls. If the offset from this boundary is greater than a byte, the adjusted address (doubleword address offset) will also be included in the picture. This adjusted address is printed in the indentation space immediately to the left of the first box.

Figure 61 on page 185 illustrates the redefinition of fields. The original input statements used to generate the picture are left in the example.

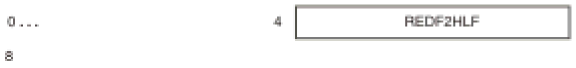
```
REDFSAM DSECT
        SPACE 1
        REDFSAM -
```



```
        REDFSAM -
        SPACE 1

* REDEFINITION SAMPLE
REDFWD   DS    F         FULL WORD IN THE MAIN PICTURE
REDF2HLF DS    2H        TWO HALF-WORDS IN THE MAIN PICTURE
REDF2BTE DS    2X        FOUR BYTES
REDF2CHR DS    2X        AND TWO CHARACTERS
         EJECT
         REDEFINITION - FIRST RE-ORIGIN
```



```
         REDEFINITION - FIRST RE-ORIGIN
         SPACE 1
REDFORG1 ORG   REDFWD   FIRST RE-ORIGIN
REDF1#F  DS    F        SAME DEFINITION AS MAIN PICTURE, NEW LABEL
REDF1#4C DS    4C       4 CHARACTERS IN FIRST RE-ORIGIN
         DS    2C       NO LABEL PICKED UP- SYNTACTICALLY DIFFERENT
         DS    2C       LABEL COPIED FROM MAIN PICTURE, SAME DEF.
         SPACE 2
         REDEFINITION - ORIGIN TO REDEFINITION, NOT MAIN PICTURE
```



```
         REDEFINITION - ORIGIN TO REDEFINITION, NOT MAIN PICTURE
         SPACE 1
REDFORG2 ORG   REDF1#4C ORIGIN TO REDEFINITION, NOT MAIN PICTURE
         DS    2H       NO MATCH AT POINT OF ORIGIN, GET MAIN PICTURE
```

Figure 61. Redefinition of Fields

Bit and Code Definition Tables: Figure 62 on page 186 illustrates the bit and code definition facility used by BLOCKMAP. The bits and codes can be in the same file as the control block definition or in an external file. The control block in SAMPLE3 will have to look externally, in CODES COPY, for the bit and code definitions. For a detailed discussion of the bit and code definition conventions used by BLOCKMAP, refer to “BLOCKMAP Conventions” on page 180.

```
SAMPLE3 DSECT
        SPACE 1
        SAMPLE3 - CODE DEFINING EXAMPLE
```

```
        SAMPLE3 - CODE DEFINING EXAMPLE
        SPACE 1
SAMBITS  DS    X         BIT SIGNIFICANT FLAG BYTE
SAMCODES DS    X         BYTE SIGNIFICANT CODE BYTE
SAMLCODE DS    2X        LONG CODE FIELD
*
         SPACE 2

BITS DEFINED IN SAMBITS  (AT HEX DISPLACEMENT: 0)

VALUE DEFINITION NAME      DESCRIPTION
----- ---------- --------  ------------------------------

X'80' 1...  .... SAMB80    DEFINE FLAG BIT 0
X'02' ....  ..1. SAMB02    DEFINE FLAG BIT 6
X'82' 1...  ..1. SAMMIX    DEFINE A SUMMATION

         SPACE 1
SAMB80   EQU   X'80'           DEFINE FLAG BIT 0
SAMB02   EQU   X'02'           DEFINE FLAG BIT 6
SAMMIX   EQU   SAMB80+SAMB02   DEFINE A SUMMATION
* END OF DEFINITION
SAMALL   EQU   X'FF'     FITS THE FORMAT, BUT NOT INCLUDED IN THE MAP
*
         EJECT

CODES DEFINED FOR SAMCODES BY CODES CBYTE
(AT HEX DISPLACEMENT: 1)

VALUE   NAME     DESCRIPTION
------- -------- --------------------------------------

X'00'   RCOK     GOOD RETURN CODE
X'FF'   RCERR    AND A LESS HAPPY ONE

         SPACE 1
         SPACE 2
```

Figure 62. Bit and Code Definition (Part 1 of 2)

```
CODES DEFINED FOR SAMCODES BY CODES STANDARD
(AT HEX DISPLACEMENT: 1)

VALUE   NAME     X'VALUE  DESCRIPTION
------- -------- -------  -----------------------------

0       NULL     X'00'    ABSENCE OF MEANING
0       ZERO     X'00'    MEANS THE QUANTITY 'NONE'

           SPACE 1
           SPACE 2

CODES DEFINED IN SAMLCODE (AT HEX DISPLACEMENT: 2)

VALUE   NAME      DESCRIPTION
------- --------  --------------------------------------

X'0010' SAMINFO   INFORMATION PROVIDED
X'0020' SAMCOMP   FUNCTION COMPLETE
X'09F0' SAMERR09  USER INPUT ERROR

         SAMPLE 1
SAMINFO  EQU    X'0010'   INFORMATION PROVIDED
SAMCOMP  EQU    X'0020'   FUNCTION COMPLETE
SAMERR09 EQU    X'09F0'   USER INPUT ERROR
SAMOTHER EQU    X'40'     NOT TO BE INCLUDED IN CODES TABLE
```

Figure 63. Bit and Code Definition (Part 2 of 2)

General Page Formatting: BLOCKMAP attempts to format the control block, with respect to the organization of groups of data, on page boundaries. Control block pictures will be divided at points that do not contain spanned records. BLOCKMAP will also attempt to maximize the number of lines per page in deciding where to divide a control block. In addition, BLOCKMAP will attempt to keep redefinitions (ORGs) with their associated data definition statements on the same page.

Because of the infinite number of combinations, it may appear that in some cases the formatting performed by BLOCKMAP is not optimal. Therefore, it is suggested that you examine the output to determine the suitability of the formatting performed.

Picture Segmentation Option: For very large control blocks, it is often desirable to break up the commented portion into smaller units. This is accomplished by including commented EJECT statements in the input file as follows:

```
     blank
EJECT     ,          NEW PICTURE
     count,
```

When this form of the EJECT statement is encountered, the current image will be completed. The data definition statements processed to that point will be combined to form part of the formatted control block, followed by this EJECT statement. Processing will then continue with the next logical statement, and a new picture will be started. This facility provides for the breaking up of the **main body** of the control block without entering redefinition. This has significance in the function of reach-back for the processing of undefined fields when in redefinition mode. Use of this option is valid for both main picture processing mode and redefinition mode.

No Picture Option: You can override the normal picture segmentation invoked by an ORG statement by specifying the NO PICTURE option as follows:

```
ORG:  label NO PICTURE and other comments
```

If this option is encountered during main picture processing, the current picture is not terminated and will be continued with the next space defining statement. This option lets you perform the following without disrupting the picture:

- Define space for operators not recognized by BLOCKMAP
- ORG back to the beginning of the reserved space
- Enter the unknown operators.

If the option is encountered on an ORG statement after the main picture is closed, the current redefinition picture is closed and processing continues without picture processing until another ORG statement is encountered.

Appendix D. Module Map Architecture (Used by ADDMAP)

PI

A module map, as referred to by the Dump Viewing Facility, is a file containing a header and a compressed form of the load map(s). The module map file is one of the inputs to the Dump Viewing Facility ADDMAP command. It can be created using the MAP command if the load map (or maps) was created by a CP loader or by the CMS LOAD command. If the load map was not created in this manner, you must create the module map manually, or by other means, if you want to append it to the dump.

The module map file is in FIXED format and has a record length of 4096 bytes. Its contents are shown in Figure 64 on page 189.

Header (1 record)
Compressed primary load map (*n* records)
Compressed secondary load map (*n* records)

Figure 64. Contents of the Module Map File

All module maps have compressed primary load maps (that is, the CMS module map); some also have compressed secondary load maps (that is, the TSAF module map).

Module Map File Header

The header is the first record of the module map file. Its format is shown in Table 14 on page 189.

Table 14. Module Map File Header Format

Byte Offset (Hexadecimal)	Field Name	Field Description
00-07	Map type	An 8-byte EBCDIC identifier representing the type of module map. If the identifier is less than 8 characters, pad this value to the right with blanks (for example, X'40').
08-17	Reserved area	16 bytes of X'00'.
18-1F	Primary load map name	An 8-byte EBCDIC identifier representing the file name of the primary load map. If the identifier is less than 8 characters, pad this value to the right with blanks (for example, X'40').
20-21	Beginning record number (primary load map)	A 2-byte right-justified hexadecimal number of the record that the compressed primary load map begins in the module map file
22-23	Beginning displacement (primary load map)	A 2-byte right-justified hexadecimal number representing the displacement of the first entry within the beginning record of the compressed primary load map.
24-25	Ending record number (primary load map)	A 2-byte right-justified hexadecimal number of the record that ends the compressed primary load map in the module map file.

Table 14. Module Map File Header Format (continued)

Byte Offset (Hexadecimal)	Field Name	Field Description
26-27	Ending displacement (primary load map)	A 2-byte right-justified hexadecimal number representing the displacement of the end of the last entry within the ending record of the compressed primary load map.
28-2F	Reserved area	8 bytes of X'00'.
30-37	Secondary load map name	An 8-byte EBCDIC identifier representing the file name of the secondary load map. If the identifier is less than 8 characters, pad this value to the right with blanks (that is, X'40'). If there is no secondary load map, fill this with 8 bytes of X'00'.
38-39	Beginning record number (secondary load map)	A 2-byte right-justified hexadecimal number of the record that begins the first entry of the compressed secondary load map in the module map file. If there is no secondary load map, fill this with 2 bytes of X'00'.
3A-3B	Beginning displacement (secondary load map)	A 2-byte right-justified hexadecimal number representing the displacement of the first entry within the beginning record of the compressed secondary load map. If there is no secondary load map, fill this with 2 bytes of X'00'.
3C-3D	Ending record number (secondary load map)	A 2-byte right-justified hexadecimal number of the record that ends the compressed secondary load map in the module map file. If there is no secondary load map, fill this with 2 bytes of X'00'.
3E-3F	Ending displacement (secondary load map)	A 2-byte right-justified hexadecimal number representing the displacement of the last entry in the ending record within the compressed secondary load map. If there is no secondary load map, fill this with 2 bytes of X'00'.
40-FFF	Reserved area	Fill with X'00' to the end.

Compressed Load Maps

Primary and secondary load maps can span any amount of records. Each compressed load map consists of contiguous entries ended by the trailer. Each entry must start on a 16-byte boundary. From the trailer to the end of the record are hexadecimal zeros. Figure 65 on page 191 shows compressed primary and secondary load map structures.

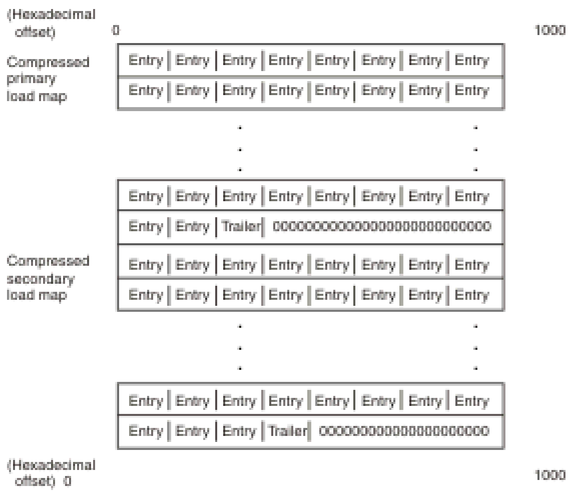


Figure 65. Compressed Load Map Structure

Table 15. Entry Format

Byte Offset (Hexadecimal)	Field Name	Field Description
00-07	Entry name	An 8-byte EBCDIC name of the module or entry point. If the name is less than 8 characters in length, pad this value to the right with blanks (that is, X'40').
08-0B	Entry address	A 4-byte hexadecimal number representing the beginning address of the entry point or module
0C	Type flag	A 1-byte hexadecimal number representing a flag to indicate whether this is an entry point or a module: **X'00'** Module **X'80'** Entry point
0D-0F	Entry size	A 3-byte hexadecimal right-justified number representing the size of the module (only valid if the type flag field is X'00').

Table 16. Trailer Format

Byte Offset (Hexadecimal)	Field Name	Field Contents
00-07	Field 1	X'FFFFFFFFFFFFFFFF'.
08-0B	Field 2	X'7FFFFFFF'.

PI end

Notices

Programming Interface Information

This book primarily documents information that is NOT intended to be used as Programming Interfaces of z/VM.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/VM. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

PI

<...Programming Interface information...>

PI end

Trademarks

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

- The section entitled **IBM Websites** at IBM Privacy Statement (https://www.ibm.com/privacy)
- Cookies and Similar Technologies (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see *z/VM: General Information*.

Where to Get z/VM Information

The current z/VM product documentation is available in IBM Documentation - z/VM (https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- *z/VM: License Information*, GI13-4377
- *z/VM: General Information*, GC24-6286

Installation, Migration, and Service

- *z/VM: Installation Guide*, GC24-6292
- *z/VM: Migration Guide*, GC24-6294
- *z/VM: Service Guide*, GC24-6325
- *z/VM: VMSES/E Introduction and Reference*, GC24-6336

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6261
- *z/VM: CMS Planning and Administration*, SC24-6264
- *z/VM: Connectivity*, SC24-6267
- *z/VM: CP Planning and Administration*, SC24-6271
- *z/VM: Getting Started with Linux on IBM Z*, SC24-6287
- *z/VM: Group Control System*, SC24-6289
- *z/VM: I/O Configuration*, SC24-6291
- *z/VM: Running Guest Operating Systems*, SC24-6321
- *z/VM: Saved Segments Planning and Administration*, SC24-6322
- *z/VM: Secure Configuration Guide*, SC24-6323

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6269
- *z/VM: Performance*, SC24-6301

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6260
- *z/VM: CMS Primer*, SC24-6265
- *z/VM: CMS User's Guide*, SC24-6266
- *z/VM: CP Commands and Utilities Reference*, SC24-6268

- *z/VM: System Operation*, SC24-6326
- *z/VM: Virtual Machine Operation*, SC24-6334
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6337
- *z/VM: XEDIT User's Guide*, SC24-6338

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6256
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6257
- *z/VM: CMS Application Multitasking*, SC24-6258
- *z/VM: CMS Callable Services Reference*, SC24-6259
- *z/VM: CMS Macros and Functions Reference*, SC24-6262
- *z/VM: CMS Pipelines User's Guide and Reference*, SC24-6252
- *z/VM: CP Programming Services*, SC24-6272
- *z/VM: CPI Communications User's Guide*, SC24-6273
- *z/VM: ESA/XC Principles of Operation*, SC24-6285
- *z/VM: Language Environment User's Guide*, SC24-6293
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6295
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6296
- *z/VM: OpenExtensions Commands Reference*, SC24-6297
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6298
- *z/VM: OpenExtensions User's Guide*, SC24-6299
- *z/VM: Program Management Binder for CMS*, SC24-6304
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6313
- *z/VM: REXX/VM Reference*, SC24-6314
- *z/VM: REXX/VM User's Guide*, SC24-6315
- *z/VM: Systems Management Application Programming*, SC24-6327
- *z/VM: z/Architecture Extended Configuration (z/XC) Principles of Operation*, SC27-4940

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6255
- *z/VM: CP Messages and Codes*, GC24-6270
- *z/VM: Diagnosis Guide*, GC24-6280
- *z/VM: Dump Viewing Facility*, GC24-6284
- *z/VM: Other Components Messages and Codes*, GC24-6300
- *z/VM: VM Dump Tool*, GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- *z/VM: DFSMS/VM Customization*, SC24-6274
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6275
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6276
- *z/VM: DFSMS/VM Planning Guide*, SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/ioa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See z/VM Performance Data Pump.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See Data Pump Messages.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- *z/VM: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Related Products

XL C++ for z/VM

- *XL C/C++ for z/VM: Runtime Library Reference*, SC09-7624
- *XL C/C++ for z/VM: User's Guide*, SC09-7625

z/OS

IBM Documentation - z/OS (https://www.ibm.com/docs/en/zos)

Index

Product Number: 5741-A09

Printed in USA

GC24-6284-74