z/VM

VM Dump Tool

Version 6 Release 3

GC24-6242-03

Note:
Before using this information and the product it supports, read the information in "Notices" on page 257.

This edition applies to version 6, release 3, modification 0 of the IBM z/VM (product number 5741-A07) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces GC24–6242–02.

Contents

Figures

Tables

About This Document

This document describes the IBM® VM Dump Tool, a dump viewing program for z/VM® CP abend, SNAP, VMDUMP, and stand-alone dumps. Utilizing an easy to use, high performance, XEDIT-based full-screen interface, it provides functions to format and display storage and control blocks, follow chains, and locate single or multiple occurrences of a string.

Intended Audience

This document is for system programmers who are pursuing a problem prior to sending it to the IBM Support Center, or working with them to pursue a problem. 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.

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

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 xii.

Table 1. Examples of Syntax Diagram Conventions

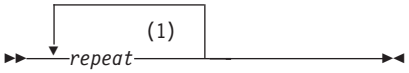
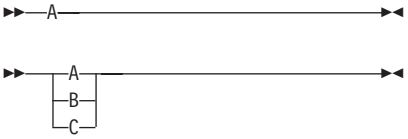
Syntax Diagram Convention	Example
Keywords and Constants A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown. In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications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

Table 1. Examples of Syntax Diagram Conventions (continued)

Syntax Diagram Convention	Example
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.	
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.

How to Use 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 VMDUMPTL commands, enter:

```
help vmdumptl menu
```

To display information about a specific VMDUMPTL subcommand (DUMPNAME in this example), enter:

```
help vmdu dumpname
```

or

```
help vmdumptl dumpname
```

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 HCQxxx010E, you can enter one of the following commands:

```
help hcqxxx010e or help hcq010e or help msg hcq010e
```

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 263.

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 Send Your Comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Go to IBM z/VM Reader's Comments (www.ibm.com/systems/z/os/zvm/zvmforms/webqs.html).

Include the following information:

- Your name
- Your email address
- The publication title and order number:
  z/VM V6.3 VM Dump Tool
  GC24-6242-03
- The topic name or page number related to your comment
- The text of your comment

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

IBM or any other organizations will use the personal information that you supply only to contact you about the issues that you submit to IBM.

If You Have a Technical Problem

Do not use the feedback methods listed above. Instead, do one of the following:

- Contact your IBM service representative.
- Contact IBM technical support.
- See IBM: z/VM Service Resources (www.ibm.com/vm/service/).
- Go to IBM Support Portal (www.ibm.com/support/entry/portal/Overview/).

Summary of Changes

This document contains terminology, maintenance, and editorial changes. Technical changes are indicated by a vertical line to the left of the change. Some product changes might be provided through service and might be available for some prior releases.

z/VM Version 6 Release 3, GC24-6242-03 (May 2015)

This edition includes changes to support product changes provided or announced after the general availability of z/VM V6.3.

Simultaneous Multithreading (SMT) Support

z/VM provides host exploitation support for simultaneous multithreading (SMT) on the IBM z13™, which enables z/VM to dispatch work on up to two threads (logical CPUs) of an IFL processor core. z/VM multithreading support is enabled only for IFL processors in a LINUX only mode or z/VM mode logical partition. If the multithreading facility is installed on the hardware and multithreading is enabled on the z/VM system, z/VM can dispatch work on an individual thread of an IFL core, allowing a core to be shared by multiple guest CPUs or z/VM Control Program tasks.

The following VM Dump Tool facilities have been updated for this support:
- CPUUSE macro
- HCQGDSPL function

Increased CPU Scalability

z/VM will support up to 64 logical processors on the z13™:

- 64 cores with multithreading disabled
- 32 cores (up to 2 threads per core) with multithreading enabled

z/VM Version 6 Release 3, GC24-6242-02

This edition includes changes to support the general availability of z/VM V6.3.

Large Memory Dump Support

The dump capability of z/VM is increased to support a real memory size up to a maximum of 1 TB.

The following VM Dump Tool subcommand has been updated for this support:
- FRAMES

z/VM Version 6 Release 2, GC24-6242-01

This edition includes changes to support the general availability of z/VM V6.2.

VM Dump Tool Enhancements

The following enhancements are included in the VM Dump Tool for this release.

- The TRACE subcommand will now support Function Related trace tables in addition to the standard CP trace table.

  - Related changes have also been made to the VMDTQRY command.
- A new macro, FRT2MAIN, has been added for Function Related trace tables.

z/VM Version 6 Release 1, GC24-6242-00

This edition includes changes or additions to support the general availability of z/VM V6.1.

Chapter 1. Introduction

Major Functions

The VM Dump Tool assists in the following tasks:

- Formatting dump data
- Interactively analyzing dump data

Formatting Dump Data

The VM Dump Tool uses dump files created by the DUMPLOAD or DUMPLD2 command. See Chapter 2, "Usage Guide," on page 9 for details.

Interactively Analyzing Dump Data

The VM Dump Tool provides a variety of subcommands and macros that allow the user to interactively locate and display dump data. Use the VM Dump Tool to do the following:

- Analyze CP system abend dumps, stand-alone dumps, SNAP dumps, and virtual machine dumps
- Display formatted data from most z/VM CP control blocks or data areas
- Display storage in hexadecimal and EBCDIC
- 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
- Determine a module entry point or displacement, given an address
- Determine an address, given a module or entry-point name
- Scroll forward or backward while viewing the contents of storage
- Format, reduce, and scroll through trace tables within a dump

Types of Dumps the VM Dump Tool Processes

You can use the VM Dump Tool to look at any of the following kinds of dumps:

- CP (control program), including stand-alone, soft abend, and snap dumps.
- A VMDUMP of a second-level CP system.

You can also use the VM Dump Tool to look at other types of dumps (from CMS, AVS, SFS, etc), although there are fewer operating system-specific subcommands and macros available.

Requirements for Using the VM Dump Tool

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

- The VM Dump Tool installed on your system. It is normally pre-installed on the MAINT 193 disk.
- The dump you want to examine that has been processed by DUMPLOAD or DUMPLD2, and resides on a disk or directory that you have access to.

- A read-write A-disk. If a VMDTMAP and VMDTBITS file is not found, the VM Dump Tool will attempt to create a new one and write it to the dump disk or (if that fails) your A-disk.

Storage Requirements

The disk storage requirements for the VM Dump Tool include space for the CP dump and the CP load map.

Table 2 shows the space requirements in cylinders for CKD/ECKD devices or blocks for FBA devices for each 16 MB of dumped storage. (For DASD types other than 3380 or 3390, you must calculate an equivalent amount of space.)

Table 2. Space Requirements in Cylinders or Blocks Per 16 MB of Dumped Storage

BLKSIZE	3380	3390	FBA	SFS
512	34	36	52,700	6588
1024	36	34	48,960	6120
2048	31	27	38,880	4860
4096	28	23	33,120	4140

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 CP takes certain types of program exceptions, a CP abend dump results. If a user enters the CP VMDUMP command in a virtual machine, a virtual machine dump results. If CP is unable to take an abend dump, you can initiate a stand-alone dump with the stand-alone dump utility. For more information, see *z/VM: CP Planning and Administration*.

Hardware-Initiated Dumps

Not all hardware errors result in a dump being taken. Some examples of hardware errors that can 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 wait 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.
Inconsistent data	A system control block or data area contains data that is inconsistent with other data or erroneous.
Address out of range	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 address out of range can be generated if the data is in error.
Incorrect instructions	An instruction is found that does not conform to the system architecture, possibly because it has been modified in main storage.

Incorrect addresses and incorrect 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

The user-initiated dumps are described below.

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 re-initializing 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). A VMDUMP is useful when CP is running in a virtual machine and a problem occurs that you wish to analyze. In order for the resulting dump to be usable by the VM Dump Tool, you must use the DUMPLOAD or DUMPLD2 command to load the dump into one or more CMS files. 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). The VM Dump Tool can not process a dump formatted for a printer. The destination of the dump is determined by using the CP SET DUMP command. For more information on the SET DUMP command, see the *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 book 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 (DVF)
- 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.

The VM Dump Tool is the supported way to look at CP dumps. It can also be used to look at other types of dumps. The Dump Viewing Facility is the supported way to look at dumps from other z/VM components or other program products. The Dump Viewing Facility can no longer be used to look at CP dumps.

Using Error Messages

One method of determining where the problem occurred is to examine any error messages that were produced. 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 VM Dump Tool also issue messages when they detect errors. Similar messages may be sent to the system operator's console. The VM Dump Tool SPCON macro can also be helpful in finding messages that are being processed but are still in buffers for the system operator or virtual machine console. (See "SPCON Macro" on page 142 for more information.) You can look at CP, CMS, and VM Dump Tool messages in the *z/VM: CP Messages and Codes* book. 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 Chapter 2, "Usage Guide," on page 9 describe this in more detail.

Using the Symptom Record to Identify Duplicate Problems

When CP terminates abnormally, or 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 VM Dump Tool SYMPTOM macro can format the symptom record for display and printing. For additional information on symptom records, see the "SYMPTOM Macro" on page 149.

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 of the VM Dump Tool

You can use the following commands with the VM Dump Tool: VMDUMPTL, DUMPLOAD, DUMPLD2, and VIEWSYM.

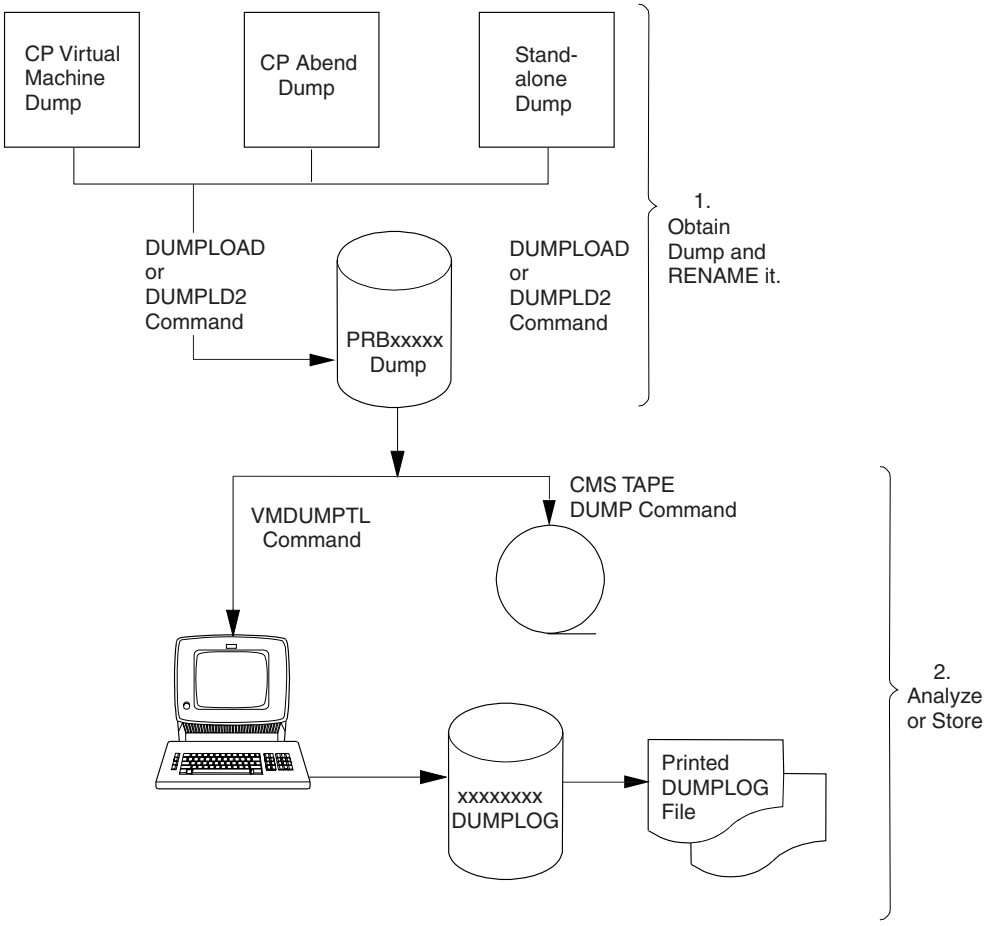


Figure 1. Command Structure for the DUMPLOAD, DUMPLD2, and VMDUMPTL Commands

1. Issue SPOOL RDR HOLD.
2. Use the DUMPLOAD or DUMPLD2 command to load the dump into one or more CMS files.
3. Use the VMDUMPTL command to interactively view CP abend, stand-alone, and CP virtual machine dumps.
4. 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 illustrates the command structure for the VMDUMPTL command. Figure 2 on page 7 illustrates the command structure for the VIEWSYM command.

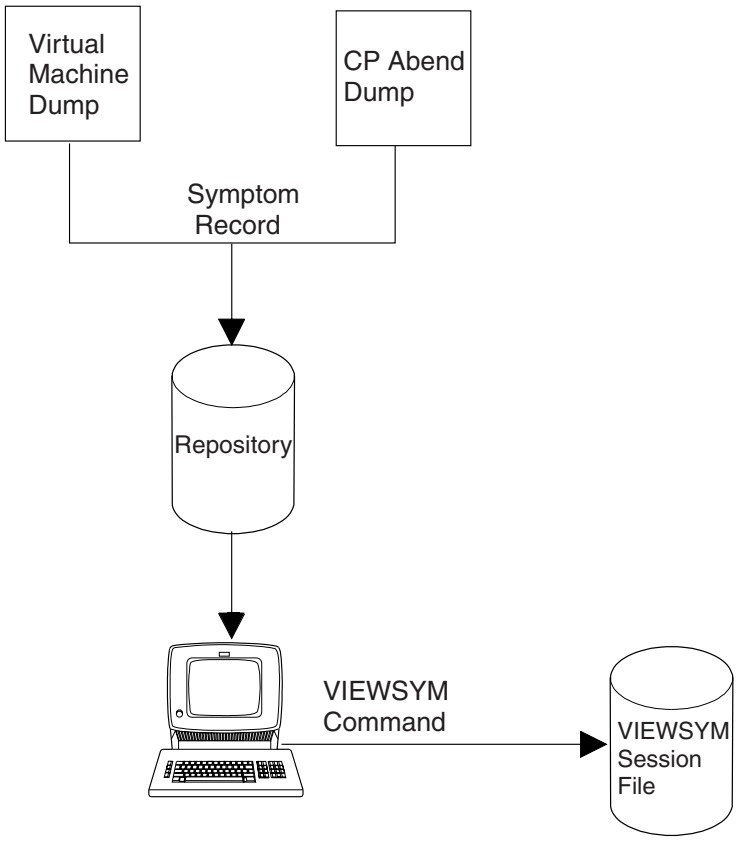


Figure 2. Command Structure for the VM Dump Tool's VIEWSYM Command

Pageable and Resident Modules

As of z/VM V5R1 all CP modules are resident in real storage rather than pageable. However since the VM Dump Tool can be used to view dumps created with a prior release of z/VM, some areas of this book, such as command responses, refer to pageable modules. For a dump created with z/VM V5R1 the term "pageable" is not found in command responses, and references to "pageable" in the text of this book can be ignored.

Servicing the VM Dump Tool

Corrective and preventive maintenance of the VM Dump Tool is performed using standard z/VM procedures. For more information on these maintenance procedures, see "Service Procedure" in *z/VM: Installation Guide*. The VM Dump Tool is serviced as part of the CP component.

Saving Dump Data to Tape

If you wish, you may copy the dump to tape as an archive, or in preparation for sending it to the IBM Support Center for further analysis.

Use the following commands to save dump data to tape:

1. Enter the DUMPLOAD or DUMPLD2 command to place the dump file(s) on your A-disk.

2. You can rename the dump file(s) to make the name reflect a PMR number, local problem log number, the abend type, or anything else to make the dump name more meaningful to you. 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
   ```

   When renaming multiple dump files of one dump, be sure to rename *all* files with the same file name.

3. Finally, you may copy the dump to tape using the CMS TAPE DUMP command or the CMS VMFPLC2 DUMP command. The format of the CMS TAPE DUMP is:

   tape dump *filename filetype filemode*

   In a similar manner, the dump can be restored with

   tape load *filename filetype filemode*

   For multiple dump files of one dump, enter the following command to pick up all the different file types, across all accessed file modes (the order of the files on tape does not matter):

   ```
   tape dump filename MDMP* *
   ```

   In a similar manner, the dump can be restored with

   ```
   tape load filename MDMP* filemode
   ```

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

Chapter 2. Usage Guide

This chapter describes the necessary steps for preparing a dump to use with the VM Dump Tool.

Preparing a Dump for Use with the VM Dump Tool

If you wish to use the VM Dump Tool 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 a R/W disk to hold the dump file.

     For the amount of storage required for dumps, see the "Storage Requirements" on page 2.
2. Issue SPOOL RDR HOLD.
3. You may use the DUMPLOAD command to load the dump into a CMS file, or the DUMPLD2 command to load the dump into multiple CMS files.
   - Issuing the DUMPLOAD command puts the dump in a CMS file on the A-disk, PRB*xxxxx* DUMP0001 A, where *xxxxx* is a number from 00000 to 99999.
   - Issuing the DUMPLD2 command allows you to choose which file modes, virtual devices or SFS directories to create the multiple CMS files on. These files have a default file name of PRB*xxxxx*, where *xxxxx* is a number from 00000 to 99999, and a file type of MDMP*yyyy*, where MDMP0001 is the first file, MDMP0002 is the second, and so on.
4. If dump files PRB00000 through PRB99999 already exist, DUMPLOAD erases all the PRB00000 files and uses the file name for the current dump.
   - DUMPLD2 will not erase any PRB00000 files; you may specify a file name of your own choice by using the 'OUTFILE' operand, or manually erase PRB*xxxxx* files to free up a PRB*xxxxx* file name.
   - 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* DUMP0001, or *filename* MDMP*xxxx*. When renaming a dump in multiple files, every MDMP*xxxx* file must be renamed. The VM Dump Tool accepts any valid CMS file name, file type, and file mode.

Viewing Dumps

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

```
vmdumptl hunguser
```

The file type defaults to DUMP0001. The VMDUMPTL command uses the first file in the search order with a file name and file type of HUNGUSER DUMP0001.

When the dump file has been accessed, the initial information from the dump is displayed. You are in an XEDIT environment ready to enter VMDUMPTL subcommands or macros on the command line.

Use the subcommands of the VMDUMPTL command to view the data in the dump file. These subcommands are explained in this book. Some of these subcommands are used in the dump analysis scenarios in this chapter. See "Scenario 1: Analyzing a CP FRE016 Abend Dump" on page 15 and "Scenario 2: Analyzing a PRG004 Abend Dump" on page 18.

Using the Session File

The VM Dump Tool 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 VM Dump Tool. This annotation feature lets you make comments within a dump session file before passing the dump on to the next level of problem determination, or to delete information that was found to be not relevant.

The dump viewing session can be filed on your A-disk (or any other R/W 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* DUMPLOG file containing the previous sessions.

Analyzing Dumps

The earlier discussion of software errors pointed out that software problems have a variety of symptoms. Two examples of these symptoms are used in this chapter's scenarios to illustrate one way of analyzing CP abend dumps, using the VM Dump Tool.

User Tailoring

You have the option to tailor the dump session further by modifying user exits that are included in the VM Dump Tool. See Chapter 5, "VM Dump Tool User Exits," on page 189 for more information.

Writing VMDUMPTL Macros

You can write your own macros to customize and automate the powers of VMDUMPTL, creating your own powerful tools to analyze dump data. For more information see Appendix B, "Writing Macros for the VM Dump Tool," on page 195.

Address Spaces and the VM Dump Tool

An address space is a consecutive sequence of virtual addresses, together with the specific transformation parameters which allow each virtual address to be associated with a byte location in storage. The sequence starts at zero and goes left to right from low address to high. A virtual address within an address space can be referenced directly by an instruction. Not all of an address space need be present in real storage when the program executes. A page can be paged out to DASD, or may not exist.

The process of deriving a real address from a virtual address is called Dynamic Address Translation. Translation of addresses within an address space to real storage addresses is handled by the hardware under control of Region, Segment and Page tables, which are built and maintained by CP. An address space is referenced by an ASCE (Address Space Control Element), which is made up of the address of the highest level translation table for that address space and some control bits. CP controls what address space(s) are available to a program by loading appropriate values into the PSW, Control Registers and Access Registers.

Many different address spaces are used within CP. The most important of these include:

- The 'real' address space, the native addresses of the real machine, which can be referenced without any translation tables.
- The CP System Execution Space, also known as the 'host logical' address space, which is used to reference CP itself and most of its control blocks. The ASCE for this address space is found in the PFXPG in field PFXSXASC.
- Other CP address spaces including System Virtual, VDISK, MDC and PTRM, as well as at least one address space for each virtual machine.

The VM Dump Tool provides three different addressing modes to make it easy to reference each of these various address spaces:

- Logical, which references the CP System Execution Address Space
- Real, which references the native addresses of the real machine
- User-defined, which can be used to reference any other address space which you want to look at in the dump.

The most important address space to the dump reader is LOGICAL, and this is normally the default addressing mode in the VM Dump Tool for CP dumps. There are two ways to reference an address space other than the default:

- use a prefix character of L, R or U on an address in a subcommand or macro, or
- use the VMDTSET ADDRESS subcommand to set the default mode to LOGICAL, REAL or USER.

Due to their function, many VM Dump Tool subcommands require a certain type of address. Such subcommands and macros do not allow a prefix character on an address, nor do they assume the default addressing mode. These address requirements are included with the description of each subcommand.

Other subcommands and macros such as DISPLAY, LOCATE and BLOCK are more general. They honor the default addressing mode and accept prefix characters on addresses. The capability and requirements of each subcommand and macro is included in its description.

The default addressing mode in a CP dump is normally LOGICAL. This means that without further indication, the VM Dump Tool will treat an address as a reference to the CP System Execution Space, translate it with the tables included in CP to a real address, and find the data in the dump file for that address. If you want to reference storage that is part of some other address space, you can either obtain the real address of that storage and reference it by real address, or you can supply an ASCE that describes the address space (a 'user defined' address space) and reference it as a 'user defined' address.

The current addressing mode can be determined by VMDTQRY ADDRESS (used here in its abbreviated form of DTQ):

```
>>> dtq address
ADDRESS  set to LOGICAL
```

This is also displayed as part of VMDTQRY ALL output:

```
>>> dtq
ABEND    set to SVC002
ABLIB    set to HCQAB540
ABSOLUTE set to OFF
ADDRESS  set to LOGICAL
ASCE     set to 00000000 00000020
BITS     set to OFF
BLKLIBS  set to HCQD8540 HCQD8530 HCQD8520 HCQB8510
DEBUG    set to OFF
DVFMACRO set to OFF
HIWORD   set to FULL
IMPDVF   set to OFF
INDENT   set to   0
LEVEL    set to 540
LINESIZE set to 72
MACLVL   set to   0
MNEMLIB  set to HCQMN540
PREFIX   set to ON
REGS     set to MAP      LONG
SEPARATE set to ON
TRACE    set to NEW ADDRESS CPU/TIME NOCODE DATA NOSLASH
    LIB  set to Default> HCQTR540
    TERM set to ADDRESS  ASCBK    ASTE     CARBK    CCTBK    COMBK
                CPEBK    DUMPCODE ESW      EXTCODE  FCTBK    FRMTE
                HASHID   IACCODE  IORBK    IPARML   IUCVB    IXBLK
                LATBK    LDEVNO   LNKBK    LOCBK    LOCK     LOCTHRED
                LSCH     MDEBK    MODULE   MSGBK    PATH     PGMBK
                PGMCODE  PTHBK    RDEV     RDEVNO   RSCH     SID
                SNABK    SUBCH    SVCCODE  SYSSVCCD TCHBK    TCHKEY
                TCHTKFMT TSKBK    VDEV     VDEVNO   VMDBK    VSCH
                WEIBK    XCRBK    XITBK
    TYPE set to APPC     CALL     CCALL    CCS      CONTROL  EXIT
                EXT      EXTERNAL FREE     I/O      IO       IUCV
                LI/O     LIO      MDC      PC       RI/O     RIO
                SCSI     SCSILOCK SENSE    SIE      SIGP     SPINLOCK
                STACK    STORAGE  VCTC     VI/O     VIO      VSTORAGE
XEDITPRE set to XEDIT
```

Every logical address has a corresponding real address. The simplest way to determine the real address is to reference the logical address, such as with a DISPLAY command, and then invoke the LASTTRAN macro which will show how it was translated.

Example 1:

The address below is that of a VMDBK. To display the first 10 bytes from this logical address:

```
>>> d 0397C000.10
_0397C000 +0000 104100A4 00000000 00000000 00000000 *...u............*
```

Use the LASTTRAN macro to display the information on how this address was translated:

```
>>> lasttran
Analysis of  00000000_0397C000 translated on ASCE 00000000_0000A007
Table type Table address     + displ  contains
---------- ----------------- -------- -----------------
Region 1st (not applicable)
Region 2nd (not applicable)
Region 3rd 00000000_0000A000 00000000 00000001_2FFF4007
Segment    00000001_2FFF4000 000001C8 00000001_2C5F6800
Page       00000001_2C5F6800 000003E0 00000001_076D2000
           (translation was successful)
```

The corresponding real address is listed as the last element on the 'Page' line above. That real address can be displayed with the DISPLAY subcommand directly by including the 'R' prefix character ahead of the address.

```
>>> d r1076D2000.10
00000001_
_076D2000 +0000 104100A4 00000000 00000000 00000000 *...u............*
```

The data displayed is the same because it is coming from the same place in the dump.

The default can be set to REAL addressing with the VMDTSET command:

```
>>> vmdtset addr real
complete

>>> dtq addr
ADDRESS  set to REAL
```

The VM Dump Tool now assumes that an address is real unless otherwise specified:

```
>>> d 1076D2000.10
00000001_
_076D2000 +0000 104100A4 00000000 00000000 00000000 *...u............*
```

You can also display this storage with the original logical address, but you must specify a prefix character of 'L' since the default is now Real.

```
>>> d L0397C000.10
_0397C000 +0000 104100A4 00000000 00000000 00000000 *...u............*
```

Example 2:

In a similar manner, it is possible to have the VM Dump Tool use an address space other than Logical or Real. This is called a User-defined address space. Use the ASCE subcommand to tell the VM Dump Tool where to find the translate tables for that address space. Use the 'U' prefix character for a user-defined address, or set the default with VMDTSET ADDRESS USER.

This example illustrates how to display location 1000 of the AVS virtual machine. To reference storage that belongs to AVS, it is necessary to find the ASCE (Address Space Control Element) for that user, which is stored in the VMDBK. The VMDBK for AVS can be found with the VMDBK subcommand:

```
>>> vmdbk avs
AVS      VMDBK at 10B01000
```

The ASCE for the address space that belongs to this user is found in the field VMDPASCE in the VMDBK. This can be displayed with the BLOCK macro:

```
>>> block vmdbk 10B01000 field vmdpasce
+0768 VMDPASCE       00000000 49D1C000  Primary Address-Space-Control
                                        Element Serialized by ASCLOCK
                                        held exclusive (all guest
                                        types) plus VMDPTIL held
                                        exclusive (for V=V guests
                                        only) Used for ESAME only
```

The ASCE subcommand is used to tell the VM Dump Tool what address space to use:

```
>>> asce 49D1C000
ASCE     set to 00000000 49D1C000
```

Now we use the 'U' prefix character to designate translation by the 'user defined' address space.

```
>>> d u1000.10
_00001000 +0000 00408A60 00000F00 000010E8 8040C1CE *. .-.......Y. A.*
```

As in previous examples, we can use LASTTRAN to display the results of the last address translation and the real address that corresponds to this user address.

```
>>> lasttran
Analysis of  00000000_00001000 translated on ASCE 00000000_49D1C000
Table type Table address     + displ  contains
---------- ----------------- -------- -----------------
Region 1st (not applicable)
Region 2nd (not applicable)
Region 3rd (not applicable)
Segment    00000000_49D1C000 00000000 00000000_4AED8800
Page       00000000_4AED8800 00000008 00000000_48950000
           (translation was successful)
```

Note that the ASCE used for this translation is the one we specified in the ASCE subcommand previously.

If we display the real address indicated above with an 'R' prefix character, we display the same storage.

```
>>> d r00000000_48950000.10
_48950000 +0000 00408A60 00000F00 000010E8 8040C1CE *. .-.......Y. A.*
```

One note on the above example is in order. Many types of dumps do not include pages which belong to users. The above example was from a stand-alone dump, specifically one that was created by the stand-alone dump program based on the HCPSADMP utility. That program creates a dump which includes everything in real storage at the time of the dump.

Example 3:

Sometimes it is necessary to look at or refer to data directly in the real storage of the machine. One example is the trace table. The TRACE subcommand displays the real address of each trace entry:

```
>>> trace for 5 one
7FCDC6A0 04:51:47 Soft Abend BVM003 at HCPBVM+596
7FCDC680 04:51:47 Return to HCPBVM+580 fr HCPGAL+8A sav 03A63A00
7FCDC640 04:51:47 /Monitor event at HCPSVC+2896
7FCDC620 04:51:47 Call from HCPBVM+580 to HCPGALSE sav 03A63A00
7FCDC600 04:51:47 Return to HCPBVM+56A fr HCPGAL+C4 sav 0372D600
```

If you try to display a trace entry in logical addressing mode, without indicating that this is a real address, the VM Dump Tool will treat the address as a logical

address and will try to translate it, with unpredictable results. Even if the translation works, the data that is displayed will not be the storage you intended.

```
>>> d 7FCDC6A0.20
HCQNID004E Error encountered on address 00000000_AD9E06E0
Error 05, Translation error, use LASTTRAN for details
```

Indicate that this is a real address by using the 'R' prefix character. The DISPLAY subcommand will now display the desired storage:

```
>>> d r7FCDC6A0.20
_7FCDC6A0 +0000 74004C52 09FC5D97 00000200 C2E5D403 *..<...)p....BVM.*
_7FCDC6B0 +0010 40E2E5C3 00020004 04040000 009A0546 * SVC............*
```

Or you can set the default to real addressing mode:

```
>>> dts addr real
complete
```

In Real addressing mode, no translation is done and the data at that real address is displayed:

```
>>> d 7FCDC6A0.20
_7FCDC6A0 +0000 74004C52 09FC5D97 00000200 C2E5D403 *..<...)p....BVM.*
_7FCDC6B0 +0010 40E2E5C3 00020004 04040000 009A0546 * SVC............*
```

Other notes:

- The default addressing mode is normally logical, but it can be set to Logical, Real or User.
- The default ASCE for User-defined addressing is to use real addresses. So unless you specify an ASCE, references to the user-defined address space are actually treated as real addresses.
- Not all of an address space may be present in real storage. If you try to display data from a page which was paged out, the subcommand or macro will fail with an error message to that effect.
- Only a limited number of subcommands and macros allow the specification of L, R or U. Many commands require their inputs to be of specific address types, so a type designation is not allowed.

Scenario 1: Analyzing a CP FRE016 Abend Dump

In this scenario, CP terminated abnormally with a FRE016 abend code and a dump. The steps that follow suggest one way this problem could be analyzed. The analysis is designed to meet three objectives: problem determination, problem source identification, and information gathering.

Step 1: Checking the Error Message and Symptom Record

Two possible methods for determining the reason for an abend are:

- Checking the error and informational messages displayed on your screen
- Using the SYMPTOM macro of the VMDUMPTL command.

Checking the Error and Informational Messages

The system operator was notified of the problem by the message:

```
HCPDMP908I  SYSTEM FAILURE ON CPU 0000, CODE - FRE016
```

HCP indicates the message is from the control program (CP). **DMP** indicates that module HCPDMP wrote the message. **908** is the message number, and **I** is the severity code. Using this information, look up the message in *z/VM: CP Messages*

and Codes or issue HELP MSG HCP908I. For this scenario, the explanation in *z/VM: CP Messages and Codes* indicates that CP has encountered a severe software failure causing a dump to be taken.

The message text also indicates that an abend has occurred. The CP abend code in the message is FRE016.

You can find the explanation for this particular abend code in the section on CP abend codes in *z/VM: CP Messages and Codes*. If you have the dump loaded, a summary of this information is available by using the VM Dump Tool DESCRIBE macro (see "DESCRIBE Macro" on page 54 for more information.) The explanation for a FRE016 code is that the control block being returned to the free storage manager has had its header or trailer, or both, overlaid.

Using the SYMPTOM Macro of VMDUMPTL

In order to use any subcommands and macros in the VM Dump Tool, you first have to load the dump you wish to view into one or more CMS files by using the DUMPLOAD or DUMPLD2 command. Then use the VM Dump Tool subcommands and macros.

In this case, we have renamed the file to 2T0O990 DUMP0001 P1. Assume you already have loaded the dump you want to examine into a CMS file, and invoked the VM Dump Tool. Enter the SYMPTOM macro to display formatted symptom record information.

An example of the output you receive if you enter the SYMPTOM macro for an FRE016 abend is shown in Figure 3.
The symptom string information is interpreted as follows:

```
>>> SYMPTOM
Symptom Record for Incident BCACAA5A C807BSYM

TOD Clock . . BCACAA5AC807B34C         Date. . . . . 03/07/05
Time Zone . . -05.00.00                Time. . . . . 10:00:55.567483
CPU model . . 2084                     Base SCP. . . 5741
CPU Serial. . 0B6F5A                   NodeID. . . . GDLVMBIG
Dump Name . . 2T0O990 DUMP0001 P1      Dump Type . . CPDUMP
Comp ID . . . 5741A05                  Ver/Rel/Mod . V05R02M0
Dump format . 64-BIT
-------------------------------------------------------------
Primary Symptom Strings
              PIDS/5741A0502           (Component ID)
              AB/SFRE016               (Abend Code)
              RIDS/FRE                 (Failing Module)
              REGS/FFFFF               (Register/PSW Info)
-------------------------------------------------------------
Section 5 Data:
              USERID DUMPED: SYSTEM
              DUMP RECEIVER: OPERATNS
              SPOOLID: 0018
-------------------------------------------------------------
Last trace entry on abending processor
7D696120 10:00:55 Abend FRE016 at HCPFRE+B2C
-------------------------------------------------------------
Abend  Description
FRE016 The control block being returned to the free storage manager has
       had its header or trailer (or both) overlaid.
```

Figure 3. Output Format of the SYMPTOM macro

- The component identifier for this product is **5741A0502**, as indicated by the **COMPONENT ID** field.

- The module that issued the abend is **FRE**, and the reason code for the abend is **016**, as indicated by the **ABEND CODE** field.
- The value in the failing module field is **FRE**. Since this is a CP DUMP, the failing module is **HCPFRE**.
- The actual point of failure is **HCPFRE+B2C**, as indicated by the last trace entry.

At this point, problem determination and problem source identification are complete. The error has been identified as a software error, and it occurred in the CP component of z/VM. Your next step is to gather more information about the abend.

Step 2: Analyzing the Trace Table

Use the VM Dump Tool to gather additional data about the circumstances of the error. You can now enter TRACE to view these entries on your display screen. You must first enter the VM Dump Tool environment by entering the VM Dump Tool VMDUMPTL command with the dump file name. The last trace entry is displayed at the top of your screen. The last trace entry has the following content in hexadecimal:

```
>>> TRACE FOR 1 HEX
7D696120 10:00:55 0200 C6D9C510 40E2E5C3 00020000 04042000 001DE72C
```

Figure 4. Last Trace Entry in the CP Trace Table with Abend Code FRE016

The trace code is 0200. The full description of this can be found in the *z/VM: Diagnosis Guide*. The other information varies by trace entry type.

By default, the VM Dump Tool TRACE subcommand interprets and formats this information so it is more easily understood. For this example, the following is displayed:

```
>>> trace for 1
7D696120 10:00:55  FRE016 Hard Abend svc 00 at HCPFRE+B2C
                   opsw 04042000_001DE72C svcilc 0002
```

Figure 5. Formatted output from VM Dump Tool TRACE subcommand

The FRE016 abend was issued by the instruction located at hexadecimal B2C bytes from the beginning of module HCPFRE.

The information in this trace entry corresponds to the primary symptom in the dump: the FRE016 abend. The next-to-last trace entry is of greater interest, because it shows what CP processing step occurred just before the abend. Now you can examine the last two trace entries as shown in Figure 6.

```
>>> trace for 2
7D696120 10:00:55  FRE016 Hard Abend svc 00 at HCPFRE+B2C
                   opsw 04042000_001DE72C svcilc 0002
7D696100 10:00:55  Release 33 dw (???) at 03BC6540 by HCPUSP+85A
                   vmdbk 0108A000 SAK1
```

Figure 6. Last two trace entries from the CP Trace Table

This shows you that the last action before the abend on this processor was that the instruction at displacement 85A in module HCPUSP returned 33 doublewords at location 03BC6540 to free storage.

The next step is to examine this storage. There are (decimal) 33 doublewords in this control block so the trailer should be found at (decimal) 264 bytes past the indicated address. Since most values used by the VM Dump Tool are in hexadecimal, this must be converted to hexadecimal, for a length of 108. The trailer is the 16 bytes immediately following the data so the length that should be displayed is 118.

The header of the control block is found in the 8 bytes immediately before the data at the address indicated. The following is a display of the header of the control block being released:

```
>>> DISPLAY 03BC6538.8
_03BC6538 +0000 00000021 6E6E6E6E *....>>>>*
```

Figure 7. Header of the Control Block being released

The following is the content of the control block itself:

```
>>> DISPLAY 03BC6540.118
_03BC6540 +0000 00C04007 019D7E40 0C000000 00800000 *.{ ...= ........*
_03BC6550 +0010 00000000 00000000 00000000 00000000 *................*
00000000_03BC6560 to 00000000_03BC663F suppressed; same as above
_03BC6640 +0100 00000000 00000000 00000000 00004C4C *..............<<*
_03BC6650 +0110 00000318 00C2E5D4 *.....BVM*
```

Figure 8. Content of the actual Control Block

The data at 108 into the control block (03BC6648) should be the trailer data (<<<<) but contains 00000000, as shown in the example. This is probably the reason that the abend was taken.

Step 3: Summarizing the Dump Analysis

You have now determined the immediate cause of the FRE016 abend. Your next step is to search for duplicate problems at your installation to make sure the problem has not already been resolved. If no duplicates are found, contact the IBM Support Center to report the problem and its symptoms. You can summarize the symptoms in this dump as follows:

1. Immediately before the FRE016 abend, HCPUSP+85A was trying to release a control block of 264 bytes.
2. HCPFRE checked the header and trailer of the block and found that the trailer had been corrupted.

Scenario 2: Analyzing a PRG004 Abend Dump

In this scenario, CP has terminated abnormally with an PRG004 abend code and a dump. The steps that follow suggest one way this problem could be analyzed. The analysis is designed to meet three objectives: problem determination, problem source identification, and information gathering.

Step 1: Checking the Error Message and Symptom Record

Two possible methods for determining the reason for an abend are:

- Checking the error and informational messages displayed on the operator's console
- Using the SYMPTOM macro of the z/VM Dump Tool.

Step 2: Checking the Symptom Record with the VM Dump Tool

The VM Dump Tool provides a set of basic information automatically when the dump is entered for the first time:

```
01 z/VM version 5 release 2.0, service level 0000 (CP 64-BIT)
02 Generated at 02/10/05 10:26:00.000000, IPLd at 02/10/05 11:12:17.551197
03 Date 02/10/05 Time 12:53:17.926602

04 CPUID = 00021524 20640000

05 CPU address is 0000    Prefix register is 0002C000   (failing)
06 CPU address is 0001    Prefix register is 00C30000
07 CPU address is 0002    Prefix register is 00C32000
08 CPU address is 0003    Prefix register is 00C34000
09 CPU address is 0004    Prefix register is 00C36000
10 CPU address is 0005    Prefix register is 00C38000
11 CPU address is 0006    Prefix register is 00C3A000
12 CPU address is 0007    Prefix register is 00C3C000

13 7FA50A80 12:53:17  Protection except at HCPGFS+896
14                    instr ST     R11,0(,R5) vmdbk 01B94000 SYSTOOL
15                    ilc 0004 int-code 0004
16                    opsw 04040000 80000000 00000000 00B2762A
16a                   exc-addr/moncode 00000000_00000000
16b                   data-exc-code 00000000 mon/per/atm id 000F0000
16c                   per-addr 00000000_000F0000

17 Summary of CP exits
18       0 Pre-defined exits found
19       0 Dynamic exits found
20       0 Diagnose exits found

21 PRG004 A protection exception occurred while CP was in control.
```

Figure 9. Initial VM Dump Tool Screen

Line 01
indicates the version, release, service level of the CP system that abended.

Line 02
shows the dates and times that the CP system was generated and IPLd.

Line 03
includes the date and time that the abend was taken.

Line 04
shows the CPU ID taken from the real machine in which the abend was taken.

Lines 05-12
indicate the CPU number, contents of the Prefix Register for each processor in the machine, and an indication of which one took the abend.

Lines 13-16c
a formatted form of the last entry in the trace table. This provides a short explanation of the abend, the module and displacement where the abend took place, and a disassembled form of the failing instruction.

Line 17-20
is a summary of the CP Exits found in the system, including the exit code, enable status, the address of the corresponding XITBK, and a short description of the exit.

Line 21
is a copy of the text from the *z/VM: CP Messages and Codes* for this abend code.

Step 3: Checking the Symptom Record for the Abend

The Symptom Record contains the technical specifics found above, plus more information about the processor, the dump itself, and the primary symptom strings from the abend.

The following is an example of the information found in the symptom record from the subject dump:

```
>>> symptom
Symptom Record for Incident BC8D6241 800C9SYM

TOD Clock . . BC8D6241800C9CA0         Date. . . . . 02/10/05
Time Zone . . -05.00.00                Time. . . . . 12:53:17.926602
CPU model . . 2064                     Base SCP. . . 5741
CPU Serial. . 021524                   NodeID. . . . GDLFST
Dump Name . . 2T00827 DUMP0001 P1      Dump Type . . CPDUMP
Comp ID . . . 5741A05                  Ver/Rel/Mod . V05R02M0
Dump format . 64-BIT
-------------------------------------------------------------
Primary Symptom Strings
              PIDS/5741A0502           (Component ID)
              AB/SPRG004               (Abend Code)
              REGS/07004               (Register/PSW Info)
-------------------------------------------------------------
Section 5 Data:
              USERID DUMPED: SYSTEM
              DUMP RECEIVER: OPERATNS
              SPOOLID: 0002
-------------------------------------------------------------
Last trace entry on abending processor
7FA50A80 12:53:17  Protection except at HCPGFS+896
                   instr ST     R11,0(,R5) vmdbk 01B94000 SYSTOOL
                   ilc 0004 int-code 0004
                   opsw 04040000 80000000 00000000 00B2762A
                   exc-addr/moncode 00000000_00000000
                   data-exc-code 00000000 mon/per/atm id 000F0000
                   per-addr 00000000_000F0000
-------------------------------------------------------------
Abend  Description
PRG004 A protection exception occurred while CP was in control.
```

Figure 10. Output Format of the SYMPTOM Macro

Step 4: Analyzing the Trace Table

The VM Dump Tool can help you gather additional data about the circumstances of this error.

The CP trace table contains information about activity in the system. The processing flow in CP just before the abend is in the most recent trace entries. Use the TRACE subcommand of VM Dump Tool to view these entries on your display screen. If you use the MERGE option, the TRACE subcommand displays information merged by time from all processors.

```
>>> trace for 1
7FA50A80 12:53:17  Protection except at HCPGFS+896
                   instr ST     R11,0(,R5) vmdbk 01B94000 SYSTOOL
                   ilc 0004 int-code 0004
                   opsw 04040000 80000000 00000000 00B2762A
                   exc-addr/moncode 00000000_00000000
                   data-exc-code 00000000 mon/per/atm id 000F0000
                   per-addr 00000000_000F0000
```

Figure 11. Last Trace Entry in the CP Trace Table with Abend PRG004

This is the trace entry for the CP abend. You can see that the information in the entry corresponds to the primary symptom in the dump: the PRG004 abend.

The instruction indicated in the trace entry stores the low order half of General Register 11 into the address found in General Register 5. But to know how much of the address is significant, we must first determine what addressing mode was set.

In the Program Old PSW (OPSW in the trace output above), the low order bit of the first word, and the high order bit of the second word of the PSW indicate addressing mode. 08 indicates 31-bit addressing mode (18 would indicate 64-bit addressing mode). When instructions are executed in 31-bit addressing mode, the the high order half of the base register is ignored.

The next step is to find the address that was in the low order half of General Register 5.

```
>>> dts reg map
complete
>>> greg
R0 AAAAAAAA_00000000                R8 AAAAAAAA_00D8B748
R1 AAAAAAAA_00000004                R9 AAAAAAAA_00000000
R2 AAAAAAAA_00BDEDF0 HCPVCOAT       RA AAAAAAAA_3349BC08
R3 AAAAAAAA_80B27626 HCPGFS+896     RB AAAAAAAA_01B94000
R4 AAAAAAAA_0000003C                RC AAAAAAAA_00B26D90 HCPGFS
R5 AAAAAAAA_00000000                RD AAAAAAAA_3C775C00
R6 AAAAAAAA_33485E68                RE AAAAAAAA_80B27546 HCPGFS+7B6
R7 AAAAAAAA_3C4744D8                RF AAAAAAAA_80386E98 HCPSTKGT
```

Figure 12. Contents of the General Registers

DTS is an abbreviation for VMDTSET. The VMDTSET REG MAP subcommand causes the GREGS subcommand to format the output vertically instead of horizontally, label each value with the register number, and resolve each address to a module name (if possible).

The low order half of R5 contains 00000000, so the STore instruction would have modified protected low storage, which is why the protection exception was presented.

Notice that the address of the VMDBK and the user ID of the user in control can be found in the TRACE output. The VMDBK is found at 01B94000, and the userid is SYSTOOL.

```
>>> callers 3c775c00
---> CPEBK at 3C775C00
CPEXFPNT 00000000 CPEXBPNT 00000000 CPEXSFQP BBBBBBBB CPEXCPRQ 0002E000
CPEXSCHC 01 CPEXCALC C0 CPEXFORM 00 CPEXRETN 003B4F80 (32-bit)
R0 00000000                    R8 00D8B748
R1 00000004                    R9 00000000
R2 80B2776E HCPGFS+9DE         RA 3349BC08
R3 80B27502 HCPGFS+772         RB 01B94000
R4 334756E8                    RC 00B2B640 HCPGRF
R5 00000000                    RD 00B2C640 HCPGRF+1000
R6 33485E68                    RE 80B2CE44 HCPGRF+1804
R7 80B275E6 HCPGFS+856         RF 00B27048 HCPGFSIO
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *.....................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *.....................*
---> R13 has the wrong boundary to be a CPEBK
```

Figure 13. Calling Sequence

General Register 13 usually contains the address of the current save area, and can be used with the CALLERS macro to display the calling sequence of how control got to the point of failure. In this case, there is only one save area involved. That save area shows that HCPGFSIO was called from GRF+1804.

Step 5: Summarizing the Dump Analysis

By examining the last trace entry, the old PSW and the registers, you have found the immediate reason for the PRG004. You can summarize the diagnostic information in the dump as follows:

- CP terminated with a PRG004 abend.
- The failing instruction was 'ST R11,0(,R5)' at HCPGFS+896.
- The system was in 31-bit addressing mode.
- The low order half of R5 contained 00000000, so the protection exception was caused by an attempt to store into protected low storage.
- HCPGFS was entered at entry point HCPGFSIO.
- GFSIO was called from HCPGRF+1804.
- The failing user was SYSTOOL.

You now have enough information to begin searching for duplicate problems at your installation. If no duplicates are found, your next step is to contact the IBM Support Center to report the problem and its symptoms.

Chapter 3. VM Dump Tool Subcommands and Macros

This chapter describes the formats and operands of the VM Dump Tool subcommands and macros. Use VMDUMPTL, a VM utility described in "VMDUMPTL Command" on page 24, to invoke the dump tool prior to issuing these subcommands and macros.

A VM Dump Tool subcommand is a command that is valid only in the environment of the VM Dump Tool. A VM Dump Tool macro is a procedures language program (such as REXX), which issues VM Dump Tool subcommands.

VM Dump Tool subcommands and macros are generally issued from the VM Dump Tool command line, though they may be issued from a procedures language environment, such as REXX. The VM Dump Tool sets up a subcommand environment named VMDUMPTL to process subcommands that are issued from a procedures language environment.

See the *z/VM: REXX/VM Reference* chapter that describes "General Concepts -- Issuing Subcommands from Your Program" for details on how to invoke VM Dump Tool subcommands from a procedures language program (such as REXX). Also see the chapter that describes "System Interfaces -- Calls Originating from a Clause That is an Expression" for details on how REXX passes XEDIT subcommands to the VMDUMPTL SUBCOM.

VM Dump Tool subcommands and macros in this book are listed in alphabetical order for easy reference. Each subcommand and macro description includes the format and description of operands and, where applicable, usage notes, notes for macro writers, and examples.

Once you are viewing the dump, you may enter commands on the command line. The VM Dump Tool handles those subcommands that it recognizes and passes other inputs to XEDIT (which, in turn, may pass them to CMS, which may pass them on to CP).

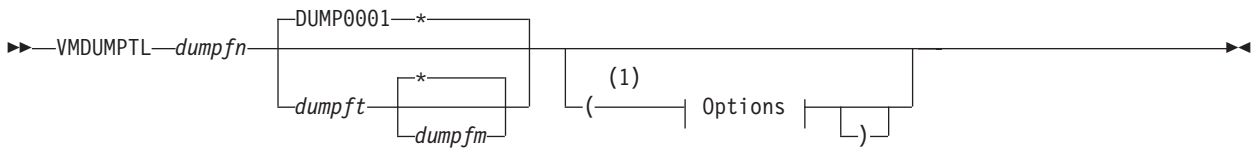
Some VM Dump Tool subcommands conflict with XEDIT, CMS, and CP commands; SET, QUERY, and DISPLAY are examples of these conflicting XEDIT commands. If SET and QUERY do not recognize the second parameter, these subcommands pass the command to XEDIT, CMS, and CP. To reduce conflicts in the future, the VM Dump Tool has introduced the VMDTSET and VMDTQRY subcommands, which are now preferred. The older SET and QUERY subcommands will continue to work and continue to be supported. New options will be added only to VMDTSET and VMDTQRY.

There are ways to force an input string to be handled by a specified level of the system:

- A prefix character or string will force a string to be passed to XEDIT for interpretation. The default is the string XEDIT, which can be changed using the SET XEDITPRE subcommand. See "SET Subcommand" on page 130 for more information.
- The VM Dump Tool CMS subcommand will force it to pass the input string to CMS to be interpreted as a CMS command.

- Similarly, the VM Dump Tool CP subcommand will force it to pass the input string to CP to be interpreted as a CP command.

VMDUMPTL Command



Options:

Notes:

1 You can enter Options in any order.

Purpose

The VM Dump Tool (VMDUMPTL) enables you to view a CP abend, stand-alone, or SNAP dump produced from a z/VM CP system. You can also use VMDUMPTL to view a non-CP dump. However, subcommands and macros that are unique to CP will not be available.

Operands

dumpfn
specifies the file name of the dump file to be processed.

dumpft
specifies the file type of the dump file to be processed. If omitted, the default file type is DUMP0001 and the dump is assumed to be a single file dump. To be processed as a multiple file dump, the file type must be specified as "MDMP*xxxx*", where *xxxx* is any number from 0000 to 9999. To use the VMDUMPTL on a multiple file dump, VMDUMPTL must be invoked against the first multiple dump file, of file type MDMP0001.

dumpfm
specifies the file mode of the dump file to be processed. If omitted, the default file mode is *. The normal CMS search order will be used to locate the dump file on any accessed disk.

PROFile *filename*
specifies that the file "*filename* XEDIT" profile should be executed when the VM Dump Tool is entered, instead of the default. The preferred user profile name is PROFILE VMDT. If not found, the VM Dump Tool will also look for VMDTPROF VMDT or VMDTPROF XEDIT (in that order).

RELease *relnum*
specifies the release and release number of z/VM that was dumped. The default is to dynamically determine the release from the dump itself so this

parameter is not normally used. If the VM Dump Tool appears to have not read the release of the dump correctly from the dump, then this parameter can be used to force it to interpret the dump at a given level.

The valid values for the release number are in the form *vrm* where:
- *v* is the version of z/VM
- *rm* is the release and modification level of z/VM

For example, the value for z/VM version 5 release 1.0 is 510. Note that the earliest supported release is z/VM version 3 release 1 (310).

The VM Dump Tool runs under z/VM CMS release 3.1.0 or later. It is strongly recommended that you have 64M or more of storage when you run the VM Dump Tool. The VM Dump Tool uses an XEDIT-based full-screen interface. All subcommands entered, and the VM Dump Tool output from those subcommands, are kept in a DUMPLOG file. With too little storage, XEDIT may run out of storage when you use VM Dump Tool subcommands that produce a large amount of output.

The VM Dump Tool may be invoked against a single CMS dump file or one dump which is stored in multiple CMS files. The VM Dump Tool ascertains if the dump is a single file or multiple file by the file type – multiple files must have a file type of the form "MDMP*xxxx*", whereas a single file dump can have any file type (the default file type for a single file dump is 'DUMP0001'). All multiple files must be available in order to view the dump, but may span across multiple file modes. One or more of the file modes may also be an SFS directory.

When you process a dump for the first time, the VM Dump Tool creates a map for the dump. The map file has the same file name as the dump file, with a file type of VMDTMAP. If the CP symbol table is found, it is used to generate the map file. If it is not found and the dump is marked version 5, release 1.0 or earlier, the VM Dump Tool scans CP storage and extracts the map information from the module headers that it finds in the dump. If the disk containing the dump is accessed R/W, the map is written on that disk as a separate file. If the dump is accessed on a R/O disk or some other error occurs, the map is written to your A-disk. (If an error occurs because, for example, your A-disk is full, the map info is not available for this dump session.)

In a similar fashion a VMDTBITS file can be created for certain types of dumps. This file is written to disk using the same conventions for the VMDTMAP file.

To use the VM Dump Tool on a dump, enter:

```
VMDUMPTL dumpfn
```

where *dumpfn* is the file name of the dump file.

The default values of the function keys are set up automatically by the VM Dump Tool when a dump is entered. You may override these by setting new values in the user profile. The default user profile is VMDTPROF XEDIT. (See the PROFILE operand mentioned previously.) You can also set colors, etc., by adding the necessary XEDIT subcommands to the VM Dump Tool user profile VMDTPROF XEDIT.

The Function keys are set as follows, by default:

F1 help for the VM Dump Tool.

F2 continuous input at the line containing the cursor. This is good for inserting your own comments into the dump.

F3 file. This saves the VM Dump Tool session on your 'A' disk so that the next time you use the VM Dump Tool on the same dump, you pick up where you left off.

F4 scroll back one subcommand. This causes the VM Dump Tool to back up to the previous CP Dump Tool command.

F5 scroll forward one subcommand. This causes the VM Dump Tool to go forward to the next VM Dump Tool subcommand.

F6 move to the next file in the XEDIT ring.

F7 back one screen. This key is interpreted by XEDIT. If it is pushed when positioned at the beginning of a file, it wraps to the end of the file.

F8 forward one screen. This key is interpreted by XEDIT. If it is pushed when positioned at the end of a file, it wraps to the beginning of the file.

F9 delete one subcommand. This is useful for deleting the output from VM Dump Tool subcommands that you no longer need in your work file. Starts at the current line (top of screen) and deletes from there to the next VM Dump Tool subcommand. After you have worked on a dump for a while, the work file can get large, and you can delete subcommands that did not supply useful information, or produced errors.

An alternative is to use the XEDIT subcommand for deleting lines from the dump session, or if you have a prefix area, XEDIT prefix macros.

F11 return the current line in the dump log to where it was before the last subcommand was entered. This is useful when you want to enter a new subcommand, see the output, and then return to where you were.

F12 retrieve the last subcommand.

In general, if you want to assign a VM Dump Tool command to a Function key, the following XEDIT command must be executed:

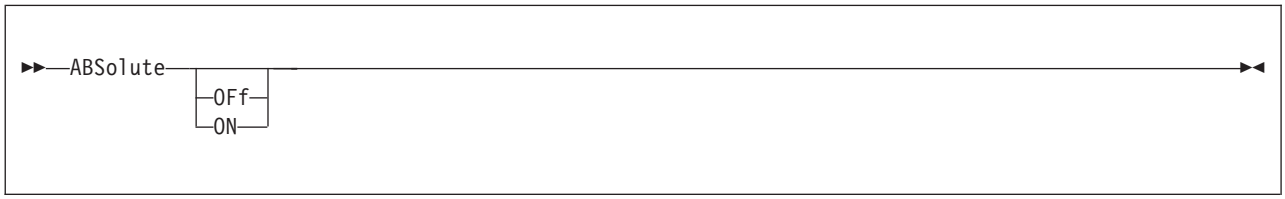
```
SET PFnn ONLY MACRO HCQREAD VMDUMPTL any VM Dump Tool command
```

Note that all Function keys should be set with the ONLY option, meaning that any data entered on the command line will be discarded when a Function key is used.

To exit from the VM Dump Tool, you can either save your session by using F3, or entering 'FILE' or you can quit by entering 'QUIT' or 'QQUIT'. These may be prefixed by the XEDIT prefix character or string.

See “VMDTSET Subcommand” on page 171 for more information on setting the XEDIT prefix string.

ABSOLUTE Subcommand



Purpose

The ABSOLUTE subcommand controls the use of absolute addresses as opposed to prefixed addresses.

Operands

(null)
causes the current setting to be displayed.

OFf
causes a reference to the prefix page on the abending processor to be treated as a real address. That is, the prefix register of the abending processor will be used in resolving such addresses. ABSOLUTE is set to OFF when the VM Dump Tool is started.

ON sets absolute addressing on. References to the prefix page on the abending processor will be treated as absolute addresses. No prefixing will be applied.

Usage Notes

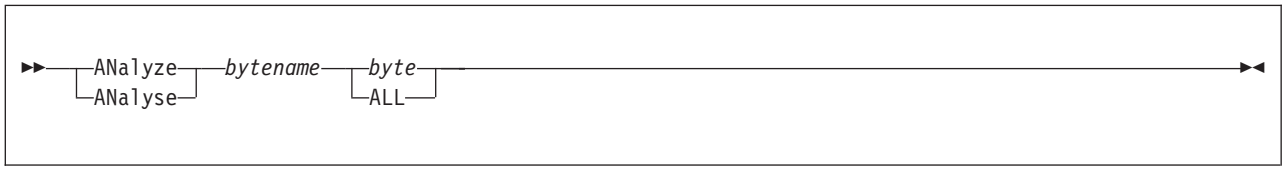
1. The ABSOLUTE subcommand is the opposite of the PREFIX subcommand. Setting ABSOLUTE ON is the same as setting PREFIX OFF. They affect the same setting and may be used interchangeably.
2. In a 64-bit mode dump, the prefix page is X'000' through X'1FFF'. In a 31-bit mode dump, the prefix page is X'000' through X'FFF'.
3. The use of the VMDTSET ABSOLUTE and VMDTQRY ABSOLUTE subcommands are preferred over the ABSOLUTE subcommand.

Examples

The following sequence shows the interaction of and the output from the ABSOLUTE and PREFIX subcommands:

```
>>> absolute
Absolute addressing off
>>> prefix
Prefixing of Page 0 is on
>>> absolute on
Absolute addressing on
>>> prefix
Prefixing of Page 0 is off
>>> prefix on
Prefixing of Page 0 is on
>>> absolute
Absolute addressing off
```

ANALYZE Subcommand



Purpose

The ANALYZE subcommand allows you to analyze various bit combinations from control blocks.

Operands

bytename
specifies the one- to eight-character name of the byte to be analyzed.

byte
is the value of the byte. It must be entered in hexadecimal and be two characters long.

ALL
specifies all the bits defined for this byte.

Usage Notes

The names of libraries currently in use are available via the VMDTQRY BLKLIBS subcommand. This list can be set or overridden by the VMDTSET BLKLIBS subcommand.

Examples

Typical use of and output from the ANALYZE subcommand:

```
>>> analyze cpexschc 64
     40 CPEXSKCR This is a stacked return
     20 CPEXSKCL This is a stacked call
     04 CPEXURGT Stack as urgent work

 >>> analyze cpexschc all
     80 CPEXNOFR Do not fret savearea on dispatch
     40 CPEXSKCR This is a stacked return
     20 CPEXSKCL This is a stacked call
     10 CPEXRTNF "Return" with no fret
     08 CPEXUCFM Stack as console function work
     04 CPEXURGT Stack as urgent work
     01 CPEXDMCO Dispatch on the master CPU only
```

AREGS Subcommand

PI

Notes:

1 The default is the abending processor.

Purpose

The AREGS subcommand displays the access registers from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

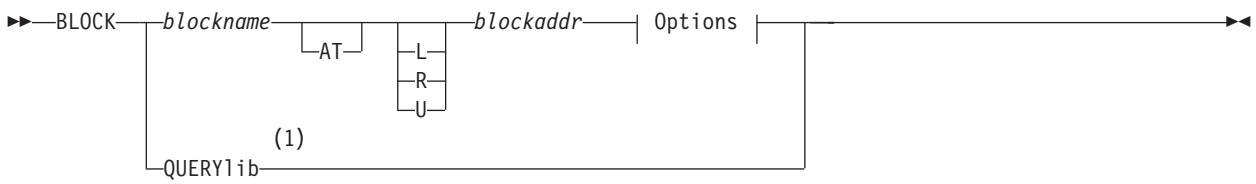
Examples

Typical use of and output from the AREGS subcommand:

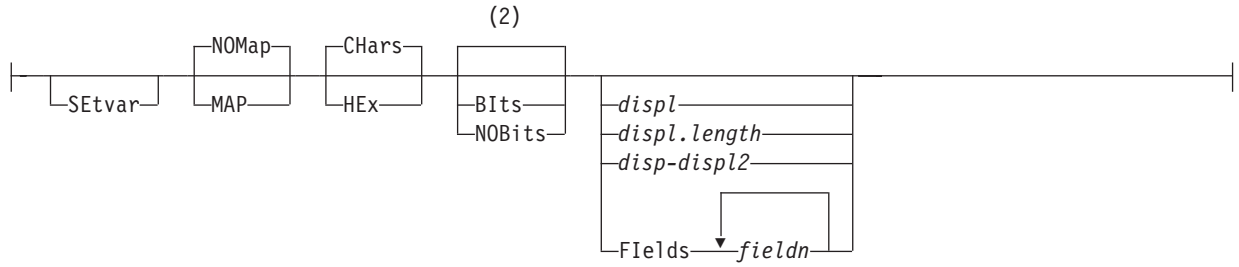
```
>>> aregs
A0-7   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

>>> aregs 2
Data for CPU 0002

A0-7   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

PI end

BLOCK Macro



Options:



Notes:

1 BLOCK QUERYlib is deprecated, use VMDTQRY BLKLIBS in its place.

2 The default is the VMDTSET BITS setting.

Purpose

The BLOCK subcommand allows you to analyze the contents of control blocks in the dump, and to determine the search sequence of VMDTDATA libraries that are used by the BLOCK and ANALYZE subcommands.

Operands

blockname
specifies the name of the block to be displayed.

AT *blockaddr*
blockaddr
is the one- to sixteen-significant digit hexadecimal address in the dump of the control block to be displayed.

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *blockaddr*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

SEtvar
specifies that each field name requested (determined by the presence or absence of the FIELDS option) is treated as a REXX variable name, and set to its value from the block being displayed. This option can be invoked only from a macro.

If CHars is specified or defaulted the data stored into each REXX variable is in raw hex (character form) as obtained from the dump. The length of the data is established by the length of the field in the control block source. If a field is requested that has a zero length, a null value is returned to the REXX variable.

If HEx is specified the value set in the variable is converted to readable hex and then stored in the variable.

NOMap
specifies that address values should not be mapped.

MAP
specifies that values found in the control block which are thought to be addresses should be mapped to a module names and displacements where possible.

BIts
specifies that definitions of bits and codes should be displayed.

NOBits
specifies that definitions of bits and codes should not be displayed.

CHars
specifies that values returned in variables should not be converted from their internal form.

HEx
specifies that values returned in variables should be converted to readable hex before being loaded into the variable. These values are directly usable as input to other VM Dump Tool subcommands and macros without any other conversion.

FIelds *fieldn*
specifies the names of the fields to be displayed (and have their values stored in REXX variables if the SETVAR option was specified).

displ
specifies the one- to three-character hexadecimal value of the only or starting displacement to be displayed from the control block.

length
specifies the one- to three-character hexadecimal length of data to be displayed from the control block.

displ2
specifies the one- to three-character hexadecimal ending displacement of data to be displayed from the control block.

QUERYlib
Displays the search sequence of VMDTDATA libraries that are used by the BLOCK and ANALYZE subcommands. The use of BLOCK QUERYlib is deprecated, the names of libraries currently in use are available via the VMDTQRY BLKLIBS subcommand. This list can be set or overridden by the VMDTSET BLKLIBS subcommand.

Usage Notes

1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.

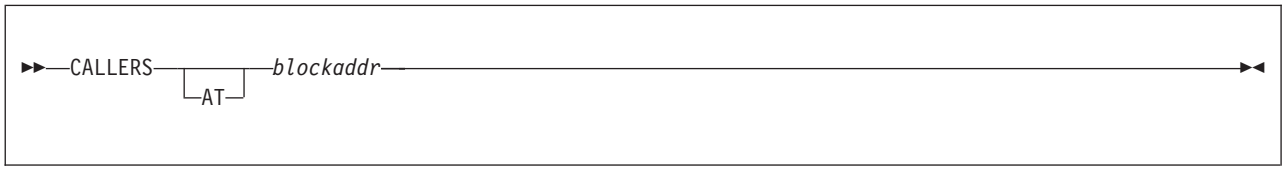
2. BLOCK chooses the data file appropriate to the release and 32– or 64–bit mode of the dump.
3. See Appendix D, "Format of Block Data," on page 219 for more information about how and where the control block definition data is stored.
4. When you use the FIELDS option, the fields are displayed in the order they appear in the original COPY file, not in the order they are listed in the command line.
5. If neither BIts nor NOBits is specified here, the value established or defaulted for VMDTSET BITS is used.

Examples

Typical use of and output from the BLOCK subcommand:

```
>>> block sfxbk 1A115698 0.6 nobits
+0000 SFXINEXT       00000000           POINTER TO NEXT SFXBK
+0004 SFXICMSK       FF162402           INTERRUPT CODES AND MASK
                                        VALUES
+0004 SFXISFMB       FF                 SOFTWARE MASK BIT NUMBER
                                        (1-31)
+0005 SFXICR0B       16                 CR0 MASK BIT NUMBER (1-31)

>>> block sfxbk 1A115698 bits
+0000 SFXINEXT       00000000           POINTER TO NEXT SFXBK
+0004 SFXICMSK       FF162402           INTERRUPT CODES AND MASK
                                        VALUES
+0004 SFXISFMB       FF                 SOFTWARE MASK BIT NUMBER
                                        (1-31)
      01 Indicates that the routine whose address is in SFXICALL is a
         fullreg routine. This bit must be turned off before using the
         address
      FE (undefined)
+0005 SFXICR0B       16                 CR0 MASK BIT NUMBER (1-31)
      16 (undefined)
+0006 SFXICODE       2402               SOFTWARE EXTERNAL INTERRUPT
                                        CODE
+0006 SFXIEXCL       24                 EXTERNAL INTERRUPTION CLASS
      24 CLASS 24 EXTERNAL INTERRUPTS (SERVICE SIGNALS)
+0007 SFXIEXCT       02                 EXTERNAL INTERRUPTION CLASS
      02 CODE X'1202' EXTERNAL CALL
      02 CODE X'2402' PVM LOGICAL DEVICE
+0008 SFXIPARM       02000006           PARAMETER TO PASS CALLED
                                        ROUTINE
+000C SFXICALL       HCPLDCEF           ADDRESS OF ROUTINE TO BE
                                        CALLED plus the last bit is
                                        used as a flag to indicate
                                        whether the routine whose
                                        address is in this field is
                                        fullreg or not
+000F SFXICAL3       D0                 Last byte of call address
      D0 (undefined)
```

CALLERS Macro



Purpose

The CALLERS macro accepts the address of a SAVBK/CPEBK/SVGBK and chains back through the saveareas of the callers of the current module, displaying and formatting all saveareas.

Operands

AT *blockaddr*
blockaddr
is the one- to eight-significant digit hexadecimal logical address in the dump of the SAVBK/CPEBK/SVGBK to be formatted.

Usage Notes

1. The macro uses the CPEBK subcommand to format the savearea block specified and any chained from it. The chain is followed until 20 blocks are formatted, a block points to itself or 0, is not in the dump, or the forward pointer is not on a boundary appropriate for a savearea.
2. The CALLERS macro interprets intervening PLX work areas to find the next CP save area, but does not display them in the output.
3. You may specify *blockaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
4. See also "FINDCPE Macro" on page 71 or the "CPEBK Subcommand" on page 45 for more detailed information on CPEBK.
5. This macro is supported only for CP dumps.

Examples

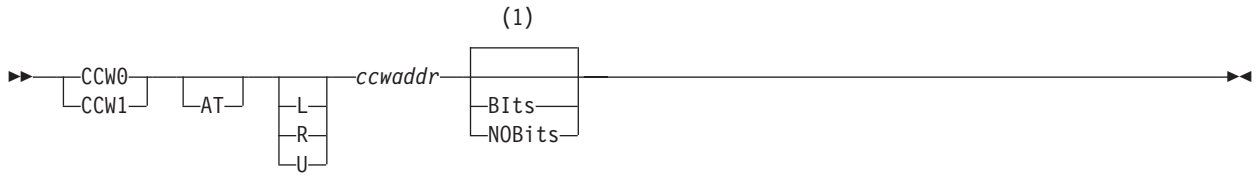
Typical use of and output from the CALLERS macro:

```
>>> callers 00000000_3FBF7D00
---> CPEBK at 3FBF7D00 in type FRMTYP frame
CPEXFPNT 00000000 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 00026000
CPEXSCHC 01 CPEXCALC 80 CPEXFORM 80 CPEXRETN 002A6B08 (32-bit)
R0 00000008                   R8 026D3C60
R1 1282E4D8                   R9 1282E4C8
R2 00000000                   RA 00335F80 HCPTTATB
R3 E2D5C1D7                   RB 36B57000
R4 C4E4D4D7                   RC 00889130 HCPCMD
R5 008901D8 HCPCOM+42A8       RD 3FBF7400
R6 00000000                   RE 8088995A HCPCMD+82A
R7 00000000                   RF 0028C3F8 HCPSNPCM
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *..................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *..................*
---> CPEBK at 3FBF7400 in type FRMTYP frame
CPEXFPNT 00000000 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 00026000
CPEXSCHC 01 CPEXCALC 80 CPEXFORM 80 CPEXRETN 002A6B08 (32-bit)
R0 00000000                   R8 026D3C60
```

CALLERS

```
R1 00000000                  R9 1282E4C8
R2 00000000                  RA 36B57000
R3 00000000                  RB 36B57000
R4 36B57000                  RC 000AE2B0 HCPCFM
R5 36B57388                  RD 00250A20 HCPPRV+1000
R6 052F7BC0                  RE 800AEA3C HCPCFM+78C
R7 000AF2B0 HCPCFM+1000      RF 008896A0 HCPCMDRT
CPEXWRK0-4 00000008 1282E4D8 00000000 00000000 00000000 *.....bUQ..........*
CPEXWRK5-9 00000000 00000000 00000000 000BCB69 77075F0A *..................*
---> R13 has the wrong boundary to be a CPEBK
```

CCW Subcommand



Notes:

1 The default is per VMDTSET BITS/NOBITS

Purpose

The CCW subcommand analyzes CCW chains in the dump.

The CCW chain is scanned and analyzed as far as possible. The logic is simple and does not consider such things as STATUS MODIFIER, therefore the entire channel program may not be found.

Operands

CCW0
specifies that the CCWs are format 0.

CCW1
specifies that the CCWs are format 1.

AT *ccwaddr*
ccwaddr
is the one- to eight-significant digit hexadecimal address in the dump of the first CCW in a chain to be analyzed.

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *ccwaddr*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

BIts
specifies that definitions of bits and codes should be displayed.

NOBits
specifies that definitions of bits and codes should not be displayed.

Usage Notes

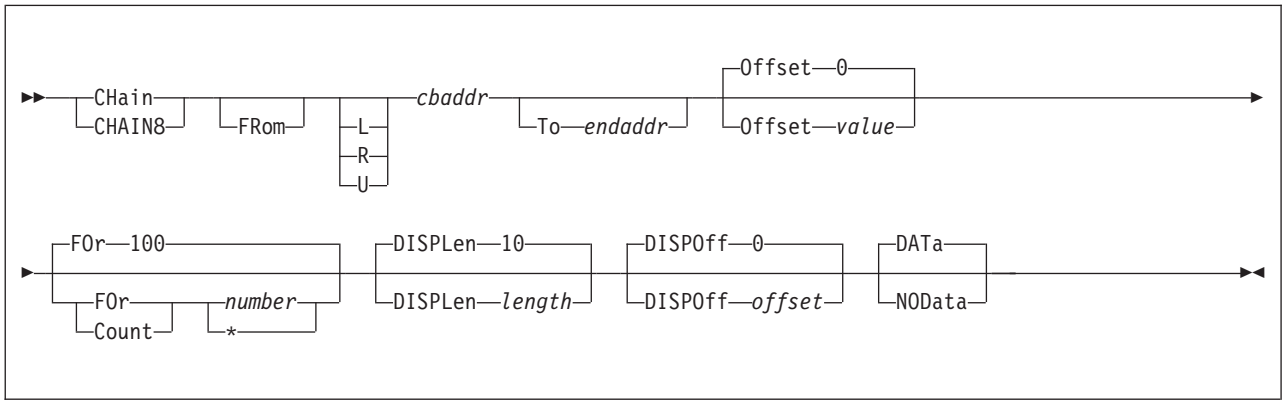
You can specify *ccwaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.

Examples

Typical output of the CCW subcommand is as follows:

```
>>> ccw1 r00000000
R00000000_00000000 CCW 00FC3000 802A4C18
R00000000_00000008 CCW FFFFFFFF FFFFFFFF
R00000000_00000010 CCW FFFFFFFF FFFFFFFF
R00000000_00000018 CCW 00000000 00000000

>>> dts bits on
complete

>>> ccw1 r00000000
R00000000_00000000 CCW 00FC3000 802A4C18
  Bits  defined in CCWFLAG          (FC)
    80 CHAIN DATA ADDRESS AND COUNT
    40 COMMAND CHAIN
    20 SUPPRESS INCORRECT LENGTH
    10 SUPPRESS INBOUND DATA TRANSFER
    08 REQUEST PC INTERRUPTION
    04 INDIRECT DATA ADDRESSING
R00000000_00000008 CCW FFFFFFFF FFFFFFFF
  Bits  defined in CCWFLAG          (FF)
    80 CHAIN DATA ADDRESS AND COUNT
    40 COMMAND CHAIN
    20 SUPPRESS INCORRECT LENGTH
    10 SUPPRESS INBOUND DATA TRANSFER
    08 REQUEST PC INTERRUPTION
    04 INDIRECT DATA ADDRESSING
    03 I/O UNDEFINED PAIR OF BITS
    02 I/O SUSPENSION/RESUMPTION
    01 I/O UNDEFINED BIT
R00000000_00000010 CCW FFFFFFFF FFFFFFFF
  Bits  defined in CCWFLAG          (FF)
    80 CHAIN DATA ADDRESS AND COUNT
    40 COMMAND CHAIN
    20 SUPPRESS INCORRECT LENGTH
    10 SUPPRESS INBOUND DATA TRANSFER
    08 REQUEST PC INTERRUPTION
    04 INDIRECT DATA ADDRESSING
    03 I/O UNDEFINED PAIR OF BITS
    02 I/O SUSPENSION/RESUMPTION
    01 I/O UNDEFINED BIT
R00000000_00000018 CCW 00000000 00000000
  Bits  defined in CCWFLAG          (00)
```

CHAIN Subcommand



Purpose

The CHAIN and CHAIN8 subcommands scan chains of linked control blocks in host storage.

Operands

cbaddr
is the one- to sixteen-significant digit hexadecimal address in the dump where the chain begins.

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *cbaddr*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

To *endaddr*
specifies that CHAIN or CHAIN8 should stop following the chain when the indicated address is found.

Offset *value*
is the one- to four-character hexadecimal value of the offset within the control block of the forward pointer to the next block. The default is 0. The maximum value is X'FFC'.

FOr | Count *number* **| ***
is a decimal number of the maximum number of blocks to display. The default is 100 blocks. The chain is considered ended when the forward pointer points to 0 or the start address for itself.

The specification of * or 0 means there is no maximum limit. Beware of using CHAIN or CHAIN8 with no limit.

DISPLen *length*
is the one- to four-character hexadecimal value of the length of data to be displayed. The default is 10. 0 means no data display. The maximum value is X'FFC'.

DISPOff *offset*
is the one- to four-character hexadecimal value of the offset where the display should start. Data from each block will be displayed, starting at this offset into the block. The default is 0. The maximum value is X'FFC'.

DATa
specifies that the contents of each block should be displayed.

NOData
specifies that the contents of each block should not be displayed.

Usage Notes

1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. To halt a long-running CHAIN or CHAIN8 command, press the PA1 key to break out of full screen mode and then enter the HI command. CHAIN or CHAIN8 will add the data accumulated thus far to the dump session, with an indication that CHAIN or CHAIN8 was terminated.
3. CHAIN loads the address of the next control block from a 4-byte field. CHAIN8 loads the address of the next control block from an 8-byte field.
4. The high order byte of an 8-byte address picked up by the CHAIN8 subcommand is ignored.

Examples

Typical use of and output from the CHAIN or CHAIN8 subcommand:

```
>>> chain 7C6C93F0
00000000_7C6C93F0 7C6C9398 00000000 00000000 00000000 *@%lq............*
00000000_7C6C9398 7C6C9340 00000000 00000000 00000000 *@%l .............*
00000000_7C6C9340 7C6C92E8 00000000 00000000 00000000 *@%kY............*
00000000_7C6C92E8 7D944AD8 7D9448A8 7D944780 7D944658 *'m¢Q'm.y'm..'m..*
00000000_7D944AD8 7D944AD8 7D944AD8 00000000 00000000 *'m¢Q'm¢Q........*
Last block points to itself
Total blocks displayed = 5

>>> chain8 017BBB10 for 10
00000000_017BBB10 00000000 801D57C0 00000000 80128480 *.......{......d.*
00000000_801D57C0 00000000 80128480 00000000 017BBB10 *......d......#..*
00000000_80128480 00000000 017BBB10 00000000 801D57C0 *.....#.........{*
Chain now loops back to start
Total blocks displayed = 3
```

CLOCKS Subcommand

Notes:

1 The default is the abending processor.

Purpose

The CLOCKS subcommand displays the clocks and related values from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

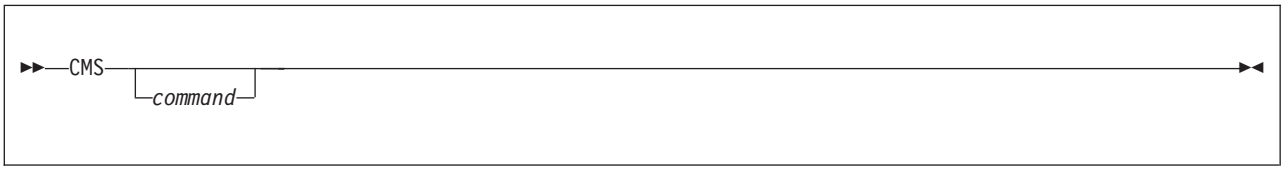
Examples

Typical output of the CLOCKS subcommand:

```
>>> clocks
TOD CLOCK B316EE8B C9CB9E00 TOD CLOCK COMP B316EE8B D0FAC000

CPU TIMER 00000000 01278200

>>> clocks 2
Data for CPU 0002

TOD CLOCK B316EE8B C9CB9E00 TOD CLOCK COMP FFFFFFFF 00000000

CPU TIMER 7FFFFFFD 291D4A00
```

See also “TODCLOCK Subcommand” on page 150 to format the contents of a clock.

CMS Subcommand



Purpose

The CMS subcommand allows you to pass a command to CMS that would normally be interpreted directly by the VM Dump Tool.

Operands

command
is the command string to be passed through to CMS.

Usage Notes

If you do not specify the name of a command you will enter CMS subset mode. See the XEDIT CMS command in *z/VM: XEDIT User's Guide* for more information.

Examples

Typical use of and output from the CMS subcommand:

```
CMS SENDFILE THIS FILE TO SOMEONE
```

CODE Subcommand

PI

Purpose

The CODE subcommand displays the ABEND code, date, time, and release from the dump symptom record.

The date displayed by the CODE subcommand is formatted when the VM Dump Tool is initialized. The date is formatted according to the setting of DATEFORMAT at entry.

See the "TODCLOCK Subcommand" on page 150 for more information. See also the "DESCRIBE Macro" on page 54 for a description of the abend code.

Usage Notes

1. The VMDTQRY ABEND subcommand can also be used to obtain the abend code from the dump.
2. The DESCRIBE macro can be used to display a summary of the meaning of an abend code.

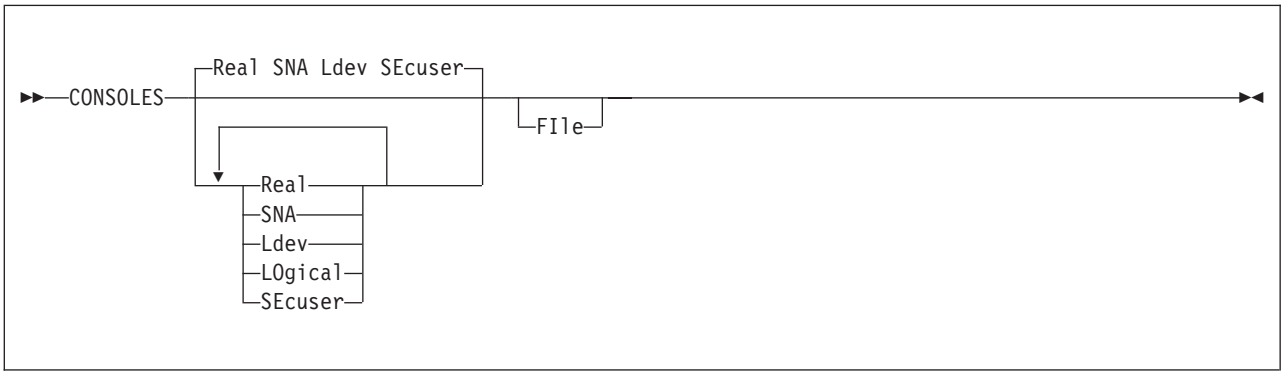
Examples

Typical output from the CODE subcommand:

```
>>> code
z/VM Abend code FRF002; Date 04/09/05   Time 00:00:38; z/VM V05R02M0
```

PI end

CONSOLES Macro



Purpose

The CONSOLES macro displays information about all or some of the user virtual consoles in the system.

Operands

Real
displays information for real device consoles

SNA
displays information for SNA device consoles

Ldev
LOgical
displays information for logical device consoles

SEcuser
displays information for secondary user device consoles

FIle
causes the output to be placed in a file called 'dumpname CONSOLES A1'. If the file already exists, the new information will be appended to the existing file.

Usage Notes

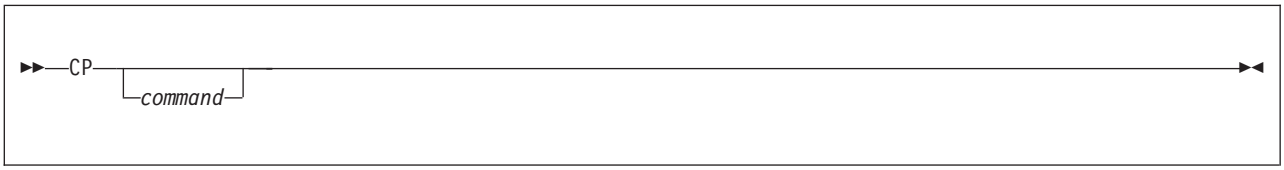
1. If none of the above options are specified, then the information will be displayed for all user virtual consoles in the system.
2. CONSOLES can generate a large amount of output; you may want to consider using the FILE option to separate this data from the current dump session. You can stop processing by using HI.
3. If FILE is not specified, the information will be placed in the current dump session.
4. Input parameters may be specified in any order.
5. This macro is supported only for CP dumps.

Examples

Typical use of and output from the CONSOLES macro:

```
>>> consoles real
OPERATOR at 770B4000 Real GRAF 3278 at 7752FDF8 dev 0081
CFTH302  at 34898000 Real GRAF 3278 at 775309D8 dev 0098
CFTHU00  at 7436E000 Real GRAF 3278 at 77530BE8 dev 0097
LOGN00A2 at 4360B000 Real GRAF 3278 at 77531350 dev 00A2
LOGN00B8 at 3509E000 Real GRAF 3278 at 77536198 dev 00B8
AE00DG02 at 044B8000 Real GRAF 3278 at 775363A8 dev 00B7
LOGN00B6 at 76009000 Real GRAF 3278 at 775365B8 dev 00B6
PGINOUT  at 4EC5A000 Real GRAF 3278 at 775339D8 dev 00A6
LOGN00A9 at 0422A000 Real GRAF 3278 at 775333A8 dev 00A9
LOGN00B4 at 7725A000 Real GRAF 3278 at 775369D8 dev 00B4
CFT3J01  at 771AF000 Real GRAF 3278 at 77533BE8 dev 00A5
CFTH303  at 73CDD000 Real GRAF 3278 at 775335B8 dev 00A8
CFT1OA   at 73717000 Real GRAF 3278 at 77531560 dev 00A1
LOGN00AC at 74FA8000 Real GRAF 3278 at 7753DBE8 dev 00AC
AJ00A    at 74FA7000 Real GRAF 3278 at 7753DDF8 dev 00AB
  15 Real Terminals
126 of 126 VMDBKs processed

>>> consoles sec
47249A   at 719C6000 Secondary user is 47249
CSERVOX1 at 44955000 Secondary user is CSERVOX
47249B   at 3489C000 Secondary user is 47249
3J01USR2 at 77259000 Secondary user is CFT3J01
3J01USR1 at 0418E000 Secondary user is CFT3J01
3J03USR2 at 4360F000 Secondary user is CFT3J03
3J03USR1 at 3D4A8000 Secondary user is CFT3J03
D250A    at 74AE3000 Secondary user is D2501
SMSMASTR at 3AE10000 Secondary user is VMTEST0
VMSERVS  at 72F45000 Secondary user is MAINT
EREP     at 76525000 Secondary user is IBMCE
  11 Secondary Users
126 of 126 VMDBKs processed
```

CP Subcommand



Purpose

The CP subcommand allows you to pass a command to CP that would normally be interpreted directly by the VM Dump Tool.

Operands

command
is the command string to be passed through to CP.

Usage Notes

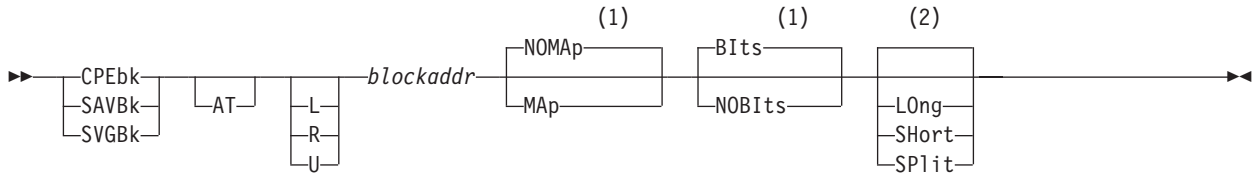
See the XEDIT CP command in *z/VM: XEDIT Commands and Macros Reference* for more information.

Examples

Typical use of and output from the CP subcommand:

```
CP MSG * HI
```

CPEBK Subcommand



Notes:

1 These definitions may be modified with VMDTSET.

2 The block should be automatically formatted according to the value found in field *xxxx*FORM. These options are valid ONLY for 64-bit dumps.

Purpose

The CPEBK subcommand displays the contents of a CPEBK, SAVBK, or SVGBK. The subcommand displays the linkage fields, registers and work areas of the block and optionally displays the meanings of bits and the mapping of registers to CP modules.

The values of MAP, NOMAP, BITS and NOBITS may be set as defaults via VMDTSET REGS MAP, VMDTSET REGS NOMAP, VMDTSET BITS ON or VMDTSET BITS OFF. The values from VMDTSET are the defaults for this subcommand.

Operands

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *blockaddr*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

AT *blockaddr*
blockaddr
is the one- to sixteen-significant digit hexadecimal address in the dump of the block to be displayed.

MAp
specifies that the values in the registers should be mapped into module names and displacements.

NOMAp
specifies that the registers should not be mapped to module names.

BIts
specifies that the meanings of bits in the CPEBK should be displayed.

NOBIts
specifies that the meanings of bits should not be displayed.

LOng
specifies that the CPEBK should be formatted in 'long' form from 64-bit

contiguous registers, independent of the format indicated in field CPEXFORM. LONG is allowed only in a 64-bit dump.

SHort
specifies that the CPEBK should be formatted in 'short' form from 32-bit contiguous registers, independent of the format indicated in field CPEXFORM. SHORT is allowed only in a 64-bit dump.

SPlit
specifies that the CPEBK should be formatted in 'long' form from 64-bit split registers, independent of the format indicated in field CPEXFORM. SPLIT is allowed only in a 64-bit dump.

Usage Notes

1. MAP and NOMAP, BITS and NOBITS can be defaulted via the VMDTSET REGS and VMDTSET BITS subcommands.
2. Use the BITS and MAP operands to get more detail from the subcommand. Use NOBITS and NOMAP to get a higher level view of the CPEBK.
3. If not otherwise specified by use of the LONG, SHORT or SPLIT operands, the format of the CPEBK in a 64-bit dump will be obtained from the CPEXFORM field in the block itself.
4. LONG, SHORT and SPLIT are not normally needed unless the value stored in the field CPEXFORM is incorrect or you wish to override it.
5. You may specify *blockaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
6. The LONG, SHORT, and SPLIT settings are **not** picked up from the VMDTSET REGS settings.
7. This subcommand is supported only for CP dumps.
8. The address specified must be on a doubleword boundary.
9. SAVBK and SVGBK are synonyms of the CPEBK subcommand.

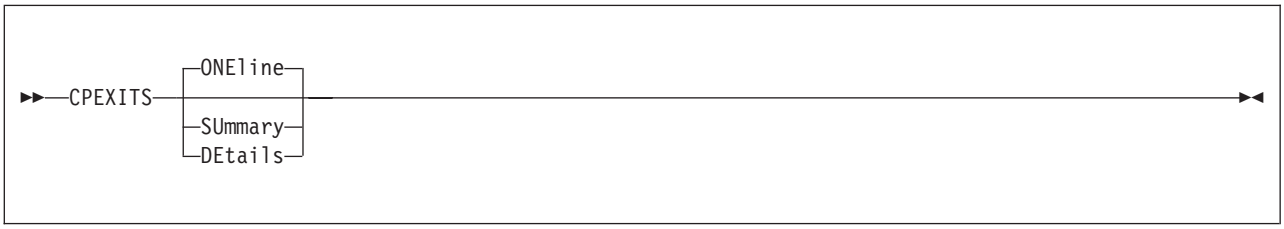
Examples

Typical use of and output from the CPEBK subcommand:

```
>>> cpebk  4416A00
CPEXFPNT 04192F80 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 01 CPEXCALC 00 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (01)
    01 Dispatch on the master CPU only
  Bits  defined in CPEXCALC         (00)
R0-7 00000005 041FE7C8 0000000C 041FE7C8 C5404040 046F4FF8 00000010 041FE640
R8-F 00000000 00000000 00000008 6F70C000 047E55F0 041FE558 847E58AE 00000010
CPEXR14 is _CDFCP+2BE
CPEXR15 is HCPPFX+10
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *....................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *....................*

>>> cpebk  04416A00 map
CPEXFPNT 04192F80 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 01 CPEXCALC 00 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (01)
    01 Dispatch on the master CPU only
  Bits  defined in CPEXCALC         (00)
R0 00000005 HCPPFX+5      R8 00000000
R1 041FE7C8               R9 00000000
R2 0000000C HCPPFX+C      RA 00000008 HCPPFX+8
R3 041FE7C8               RB 6F70C000
```

```
R4 C5404040               RC 047E55F0
R5 046F4FF8 HCPCOM+44D8   RD 041FE558
R6 00000010 HCPPFX+10     RE 847E58AE _CDFCP+2BE
R7 041FE640               RF 00000010 HCPPFX+10
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *.....................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *.....................*

>>> cpebk  04416A00 nobits
CPEXFPNT 04192F80 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 01 CPEXCALC 00 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
R0-7 00000005 041FE7C8 0000000C 041FE7C8 C5404040 046F4FF8 00000010 041FE640
R8-F 00000000 00000000 00000008 6F70C000 047E55F0 041FE558 847E58AE 00000010
CPEXR14 is _CDFCP+2BE
CPEXR15 is HCPPFX+10
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *.....................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *.....................*
```

CPEXITS Macro



Purpose

The CPEXITS macro is used to display information about pre-defined and dynamic exits and diagnose codes which have been modified by your installation.

Operands

ONEline
indicates that one line should be displayed for each instance of each type of exit.

SUmmary
indicates that only the counts of pre-defined, dynamic, and diagnose exits should be displayed.

DEtails
indicates that intermediate control block information should be displayed for all exit types. For dynamic exits, all relevant ESDBK information is displayed. A list of the relevant ICLBKs is also displayed.

Usage Notes

1. For compatibility, formerly supported CPEXITS operands continue to be supported as follows:
   - The LONG operand is equivalent to the DETAILS operand.
   - The SHORT operand is equivalent to the SUMMARY operand.
   - The FORMAT and FMT operands are equivalent to the ONELINE operand.
2. This macro is supported only for CP dumps.
3. Due to changes in CP, the CPEXITS macro supports dumps only at the level of version 5, release 2.0 or later.

Examples

Typical output from the CPEXITS macro:

```
>>> cpexits summary
Summary of CP exits
     11 Pre-defined exits found
      1 Dynamic exit found
      1 Diagnose exit found

>>> cpexits
   Exit Enab XITBK @  XCRBK @  ICLBK @  EP Name  Name
   1110 Yes  5F259D50 5F259DC8 5F9FA0C8 BHPUTLDR VMDBK, Pre-Logon
   117F Yes  5F2599B8 5F259A30 5F812058 BHPUTLAS Logon, Final Screening
   11FF Yes  5F2598E0 5F259958 5F9FA138 BHPUTLRS Logoff, Final Screening
   3FE8 Yes  5F259C78 5F259CF0 00000000 BHPSHUTD SHUTDOWN Cmd Screening
   4400 Yes  5F259808 5F259880 00000000 BHPSEPUP Sep Pg Data Customization
```

```
   F000 Yes  5F259730 5F2597A8 5F3945A8 BHPSFSUP -- User-defined exit --
   F001 Yes  5F80E100 5F394008 5F80E1E8 BHPBLOUP -- User-defined exit --
   F002 Yes  5F80E028 5F80E0A0 5F80E808 BHPLONUP -- User-defined exit --
   F003 Yes  5F259F90 5F80EAD8 00000000 BHPBOYSS -- User-defined exit --
   F004 Yes  5F259EB8 5F259F30 5F80E6B8 BHPBOYGO -- User-defined exit --
   F005 Yes  5F259BA0 5F259C18 5F394298 BHPVSQME -- User-defined exit --
     11 Pre-defined exits found
Dynamic exits
   LABK @   FileName FileType Member
   5F394618 BHPBLCPX TXTLIB   LOADPROD
      1 Dynamic exit found
Diagnose exits
   DGNBK @  EP Name  DiagName ICLBK @
   5F259E28 BHPVMINQ DIAG100  5F80ED68
      1 Diagnose exit found

>>> cpexits details
   Exit Enab XITBK @  XCRBK @  ICLBK @  EP Name  Name
   1110 Yes  5F259D50 5F259DC8 5F9FA0C8 BHPUTLDR VMDBK, Pre-Logon
   117F Yes  5F2599B8 5F259A30 5F812058 BHPUTLAS Logon, Final Screening
   11FF Yes  5F2598E0 5F259958 5F9FA138 BHPUTLRS Logoff, Final Screening
   3FE8 Yes  5F259C78 5F259CF0 00000000 BHPSHUTD SHUTDOWN Cmd Screening
   4400 Yes  5F259808 5F259880 00000000 BHPSEPUP Sep Pg Data Customization
   F000 Yes  5F259730 5F2597A8 5F3945A8 BHPSFSUP -- User-defined exit --
   F001 Yes  5F80E100 5F394008 5F80E1E8 BHPBLOUP -- User-defined exit --
   F002 Yes  5F80E028 5F80E0A0 5F80E808 BHPLONUP -- User-defined exit --
   F003 Yes  5F259F90 5F80EAD8 00000000 BHPBOYSS -- User-defined exit --
   F004 Yes  5F259EB8 5F259F30 5F80E6B8 BHPBOYGO -- User-defined exit --
   F005 Yes  5F259BA0 5F259C18 5F394298 BHPVSQME -- User-defined exit --
     11 Pre-defined exits found
Dynamic exits
   SYSCM at 00504000 (via PFXSYS)
   DLUBK at 5F63BAF0 (via SYSDLUBK)
   LABK chain (via DLULABKD)
   LABK @   FileName FileType Member   ESDBK @  EP Name  VirtAddr RealAddr
   5F394618 BHPBLCPX TXTLIB   LOADPROD
                                       5F37B3B0 BHPUTLAS 009A4058 5F37E058
                                       5F37B418 BHPUTLRS 009A40B0 5F37E0B0
                                       ...
                                       5F2BD2E0 BHPBLOUP 00998058 5F80A058
                                       5F2BD008 BHPBLO   00998000 DF80A000
      1 Dynamic exit found
Diagnose exits
   DCHBK at 5F63BA80 (via SYSDIAGS)
   DGNBK @  EP Name  DiagName ICLBK @
   5F259E28 BHPVMINQ DIAG100  5F80ED68
      1 Diagnose exit found
ICLBK chain (via DLUICL)
   ICLBK @  EP Name  VirtAddr RealAddr CPEBK @
   5F394308 BHPADF   009A1000 00000000 00000000
   5F3943E8 BHPADFQY 009A1058 00000000 00000000
   ...
   5F394228 BHPVSQ   009A0000 00000000 00000000
   5F394298 BHPVSQME 009A0058 00000000 00000000
     47 ICLBKs found
```

CPUID Subcommand

PI

Purpose

The CPUID subcommand displays the CPUID from the prefix page of the dump along with information about the processors in the configuration.

Usage Notes

1. If the leftmost byte of the displayed *cpuid* is FF, then the dump is from a virtual machine. The model number is the leftmost two bytes of the second word.
2. See the CP QUERY CPUID command in the *z/VM: CP Commands and Utilities Reference* for more information about the CPU ID.

Examples

Typical use of and output from the CPUID subcommand:

```
>>> cpuid
CPUID = 75016452 96720000

CPU address is 0000    Prefix register is 04508000   (failing)
CPU address is 0001    Prefix register is 76574000
CPU address is 0002    Prefix register is 765B0000
CPU address is 0003    Prefix register is 765EC000
CPU address is 0004    Prefix register is 7665A000
```

PI end

CPUUSE Macro

Purpose

The CPUUSE macro displays information about the use of real processors in the configuration at the time of the dump.

Usage Notes

This macro is supported only for CP dumps.

Examples

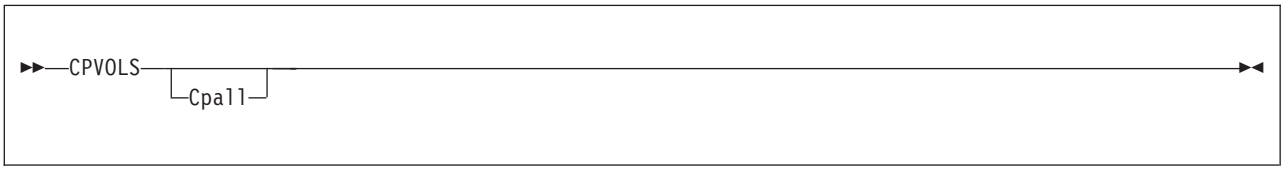
Typical output from the CPUUSE macro:

```
>>> cpuuse
CPU 0000  Master
CPU 0001  Slave
CPU 0002  Slave
CPU 0003  Slave
CPU 0004  Parked
```

When multithreading is enabled on the system in the dump, additional columns containing CPU type and core ID are displayed:

```
>>> cpuuse
CPU 0000  Master  CP   Core 0000
CPU 0002  Slave   CP   Core 0001
CPU 0004  Slave   IFL  Core 0002
CPU 0005  Slave   IFL  Core 0002
CPU 0006  Parked  IFL  Core 0003
CPU 0007  Parked  IFL  Core 0003
```

CPVOLS Macro



Purpose

The CPVOLS macro displays the CPVOL blocks for some or all volumes in the SYSCPVOL list based on the options specified. For those SYSCPVOL volumes with exposure blocks, the exposure block is also displayed.

Operands

Cpall
specifies that CPVOLs for all SYSCPVOL volumes (both online and offline) should be displayed.

Usage Notes

1. If no options are specified, CPVOLs will be displayed for only online SYSCPVOL volumes.
2. This macro is supported only for CP dumps.

Examples

Typical use of and the output for one device from the CPVOLS macro:

```
>>> cpvols
System volume entries

Volume index  0  Volid K1E434 Device number E434  Subchannel number 1AA0
  00000000_77509958 D2F1C5F4 F3F44000 77522DA0 770BB000 *K1E434.........*
  00000000_77509968 00000000 00000000 00300000 00000000 *................*
  00000000_77509978 FFFFFFFF 00000000 00000000 00000000 *................*
  00000000_77509988 00000000 00000000 *........*

  Exposure block
  00000000_043ECD50 00020100 00000000 770BA740 770BA740 *.........x...x.*
  00000000_043ECD60 00000000 00000000 00000000 00000000 *................*
  00000000_043ECD70 00000000 00000000 000009E0 0000071D *..........\....*
  00000000_043ECD80 00000000 00000000 00000000 00000000 *................*
  00000000_043ECD90 00000000 043ECC10 00000058 00000055 *................*
  00000000_043ECDA0 0000034B EE88C0A9 EE88C0A9 00000001 *.....h{z.h{z....*
  00000000_043ECDB0 776330C0 00000000 00000000 00000000 *...{............*
  00000000_043ECDC0 00000000 00000000 00000000 00000000 *................*
  00000000_043ECDD0 00000000 00000000 00000000 00000000 *................*
  00000000_043ECDE0 00000000 00000000 00000000 00000000 *................*
  00000000_043ECDF0 00000000 00000000 00000000 00000000 *................*
  00000000_043ECE00 00000000 *....*
```

CREGS Subcommand

Notes:

1 The default is the abending processor.

Purpose

The CREGS subcommand displays the control registers from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

Usage Notes

You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.

Examples

Typical use of and output from the CREGS subcommand:

```
>>> cregs
C0-7   14B1FE40 0497E07F 77814040 00000000 00000000 77814000 80000000 43B39102
C8-15  00000001 00000000 00000000 00000000 7783EB41 77631100 1F000000 00000000
```

DESCRIBE Macro

Notes:

1 The default is the abend code taken from the dump.

Purpose

The DESCRIBE macro displays the meaning of the abend code either from the dump, by default, or as specified to the macro.

Operands

code
is a z/VM abend code.

Usage Notes

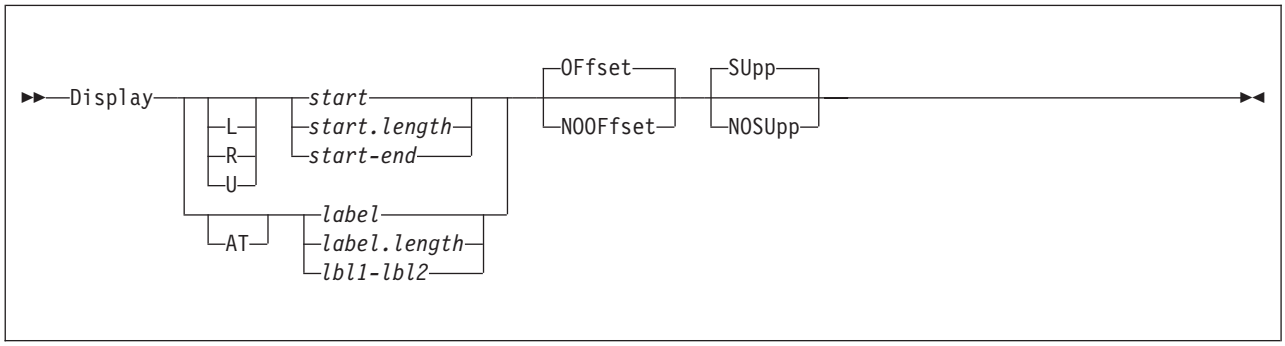
1. You may specify *code* as an underscore character ("_"). Place the cursor on an item of data in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
2. The output is a simple description of the abend code, as extracted from a data file. You can obtain the name of the data file from the VMDTQRY ABLIB subcommand. This file may be updated to add new codes or change existing ones.
3. The data in the data file has the following format:
   column 1:
   6-character CP abend code
   column 8-end:
   explanation of the abend.
4. The full definition for each abend must be contained on one line, but that line can be as long as needed.

Examples

Typical use of and output from the DESCRIBE macro:

```
>>> describe
FRE016 The control block being returned to the free storage manager has
       had its header or trailer (or both) overlaid.

>>> describe prg005
PRG005 An addressing exception occurred while CP was in control.
```

DISPLAY Subcommand



Purpose

The DISPLAY subcommand displays the contents of storage locations from within the dump.

Operands

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *start*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

start
specifies the one- to sixteen-significant character hexadecimal address of the display. *start* may be specified as a hexadecimal number or the name of a module or entry point.

length
specifies the one- to five-significant hexadecimal character length of the display. If the resulting length is 0 or if length is not specified, then approximately one screen of data will be displayed. All lengths are rounded up to generate a multiple of 4 from the start address of the display. The length of data displayed is limited to X'10000'. If a larger length is specified, it is truncated to X'10000'.

end
specifies the end address of the display. The end address must not be lower than the start address.

AT indicates that the term that follows is a label and should not be interpreted as an address.

label
specifies the name of a module or entry point in the dump to display from. A *label* can be mapped with the MAP subcommand. *label* is interpreted as a logical address.

lbl1
lbl2
specify the names of modules or entry points in the dump that define the limits of data to be displayed. The address of *lbl1* must be less than the address of *lbl2*. *lbl1* and *lbl2* are interpreted as logical addresses.

OFfset
specifies that the offset of each line from the beginning of the display should be included in the output.

NOOFfset
specifies that the offset of each line from the beginning of the display should not be included in the output.

SUpp
specifies that duplicate lines should be suppressed.

NOSUpp
specifies that duplicate lines should not be suppressed.

Usage Notes

1. The DISPLAY subcommand does not show keys associated with the storage. Use the KEY subcommand to display storage keys.
2. The DISPLAY subcommand is sensitive to the current setting of ABSOLUTE or PREFIX addressing (see the "ABSOLUTE Subcommand" on page 27 and the "PREFIX Subcommand" on page 106). When absolute addressing is off (prefixing is on), a reference to the prefix page of the machine is prefixed and redirected from the prefix register for that processor to the actual prefix page for that processor. A reference to the prefix page for the failing processor is redirected to the prefix page for the machine. The reason this is important is that the CPU runs with prefixing on. When PREFIX ON is specified, references to the failing processor's prefix page reference the same data that the CPU would without having to remember to do the prefixing step.

   Example: CPU 0 is the failing processor. The CPU 0 prefix register contains 12345000. When ABSOLUTE is set OFF (or PREFIX is set ON), a reference to storage location 100 is prefixed to reference location 12345100. A reference to 12345200 will be *reverse-prefixed* to reference storage location 00000200. In a 32-bit mode dump, the prefix page is X'1000' in length (locations X'000' through X'FFF'). In a 64-bit mode dump, the prefix page is X'2000' in length (locations X'000' through X'1FFF').
3. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.

Examples

Typical use of and output from the DISPLAY subcommand:

```
>>> display L3FBBD000.20
_3FBBD000 +0000 74009770 75F7931C 00000700 4CC3D6D4 *..p..7l.....<COM*
_3FBBD010 +0010 00000015 3F976848 36B57000 8091EE5E *.....p.......j.;*

>>> display r3FBBD000.20
_3FBBD000 +0000 74009770 75F7931C 00000700 4CC3D6D4 *..p..7l.....<COM*
_3FBBD010 +0010 00000015 3F976848 36B57000 8091EE5E *.....p.......j.;*

>>> display 80.80
_00000080 +0000 00016000 00011202 0002004C 00060016 *..-........<....*
_00000090 +0010 00000000 00030000 00000000 00000000 *................*
_000000A0 +0020 0E003101 00000000 00000000 00E94000 *.............Z .*
_000000B0 +0030 00000000 0000000C FFFF0006 026D3C60 *.............._.-*
_000000C0 +0040 00000000 00000000 FA00FA00 00000000 *................*
_000000D0 +0050 00000000 00000000 00000000 00000000 *................*
00000000_000000E0 to 00000000_000000FF suppressed; same as above
```

```
>>> display 80.80 nosupp
_00000080 +0000 00016000 00011202 0002004C 00060016 *..-........<....*
_00000090 +0010 00000000 00030000 00000000 00000000 *................*
_000000A0 +0020 0E003101 00000000 00000000 00E94000 *.............Z .*
_000000B0 +0030 00000000 0000000C FFFF0006 026D3C60 *............._.-*
_000000C0 +0040 00000000 00000000 FA00FA00 00000000 *................*
_000000D0 +0050 00000000 00000000 00000000 00000000 *................*
_000000E0 +0060 00000000 00000000 00000000 00000000 *................*
_000000F0 +0070 00000000 00000000 00000000 00000000 *................*

>>> display 80.80 noof
_00000080 00016000 00011202 0002004C 00060016   *..-........<....*
_00000090 00000000 00030000 00000000 00000000   *................*
_000000A0 0E003101 00000000 00000000 00E94000   *.............Z .*
_000000B0 00000000 0000000C FFFF0006 026D3C60   *............._.-*
_000000C0 00000000 00000000 FA00FA00 00000000   *................*
_000000D0 00000000 00000000 00000000 00000000   *................*
00000000_000000E0 to 00000000_000000FF suppressed; same as above
```

DUMPNAME Subcommand

PI

Purpose

The DUMPNAME subcommand displays the file name, file type, and file mode of the dump file you are working with. If the dump is segmented into multiple dump files of file type MDMP*xxxx*, the DUMPNAME subcommand displays the file name, file type, and file mode of the first file.

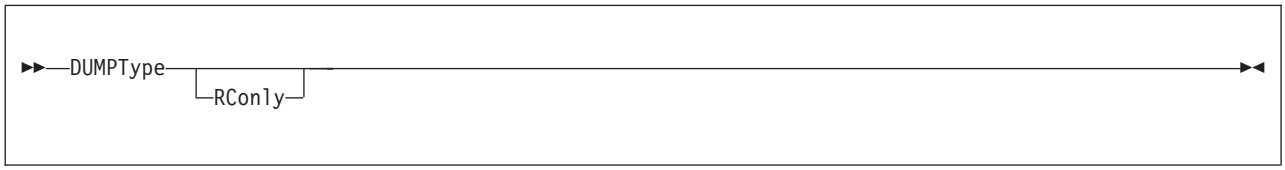
Examples

Typical use of and output from the DUMPNAME subcommand:

```
>>> dumpname
Dump file is 2T0P095 DUMP0001 P1
```

PI **end**

DUMPTYPE Subcommand



Purpose

The DUMPTYPE subcommand returns an indication of the dump type.

Operands

PI end RConly
specifies that the result is to be returned as a return code only, with no text output. For a 32-bit CP dump, the return code is 0. For a 64-bit CP dump, the return code is 1. For a non-CP dump, the return code is 2. **PI end**

Usage Notes

1. The RCONLY operand makes it very easy to determine the type of a dump from a macro.
2. When DUMPTYPE is invoked from a macro with the RCONLY parameter, all other operands on the line are ignored. If the RCONLY value is not specified correctly, or if some other incorrect parameter is found first, message 001E 'Unrecognized operand - *operand*' will be displayed and a return code of 1 will be returned to the macro. If a line of output is returned to the macro, the return code reflects the message produced and not the type of the dump.

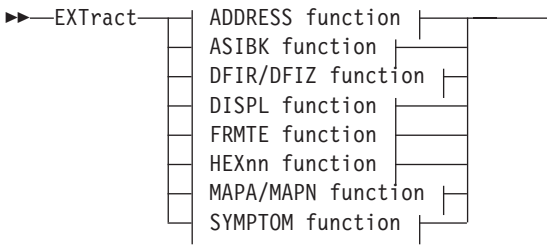
Examples

Typical use of and output from the DUMPTYPE subcommand:

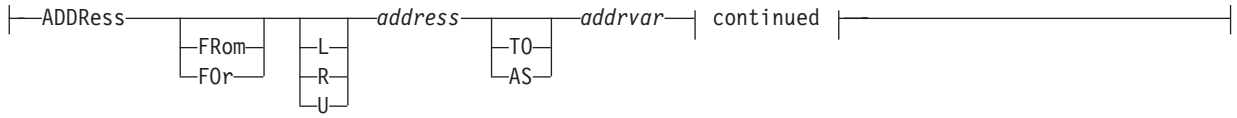
```
>>> dumptype
Dump type is CP 64-BIT
```

EXTRACT Subcommand

PI



ADDRess function:



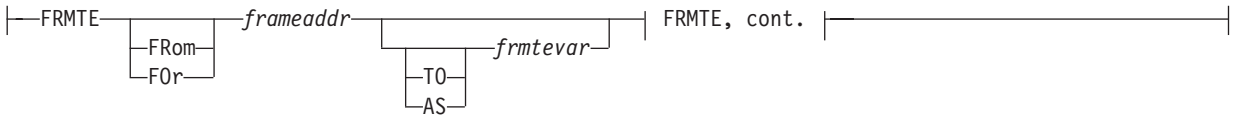
ADDRess function, continued:

ASIBK function:

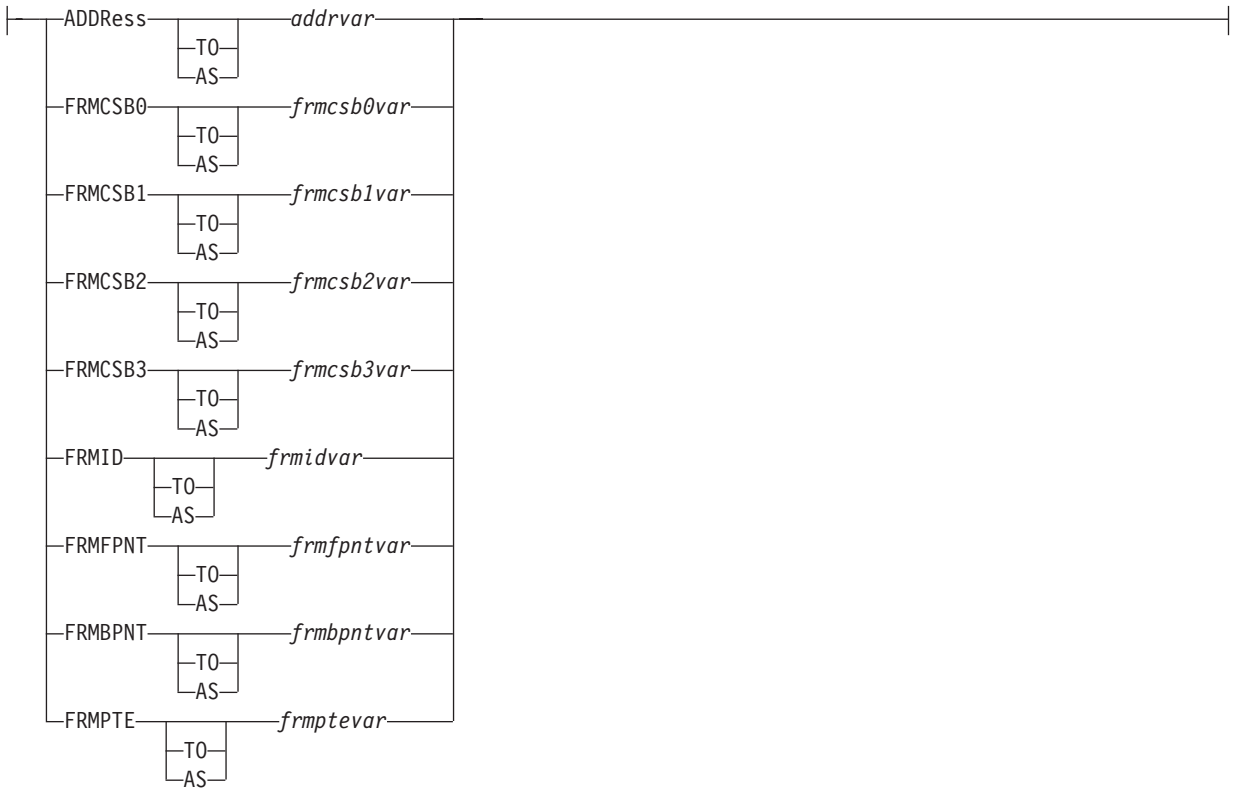
DFIR/DFIZ function:

DISPL function:

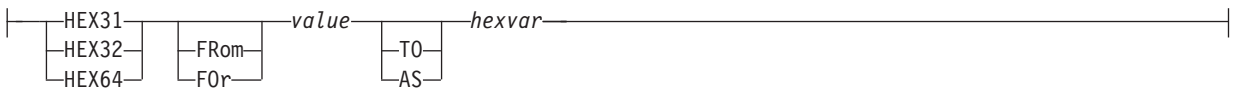
FRMTE function:



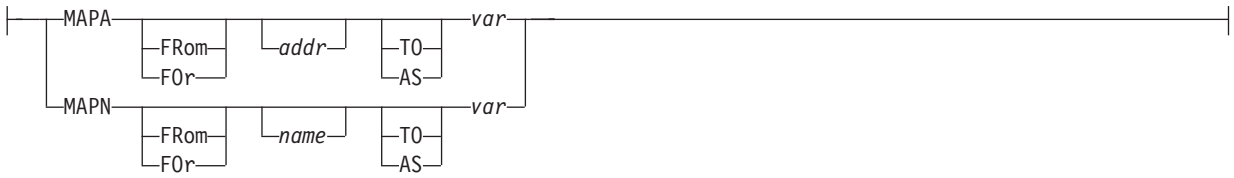
FRMTE, cont.:



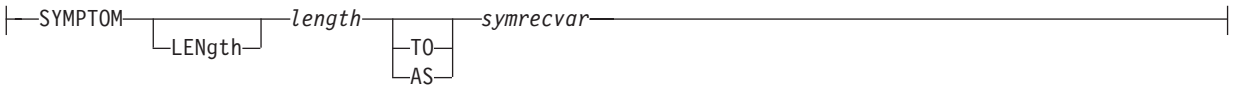
HEXnn function:



MAPA/MAPN function:



SYMPTOM function:



Purpose

Use the EXTRACT function from a macro to obtain selected information from the dump, and return the information in one or more specified variables.

Operands

ADDRess
accepts a value, checks it for being a valid hexadecimal number, translates it as specified by the prefix character or default, and returns a sixteen-digit hexadecimal real address without an underscore character.

FRom
FOr
indicates that the input address follows as the next parameter.

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *address*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

address
is the input address to be checked and translated.

TO
AS indicates that a variable name follows as the next parameter. The variable name will receive the standardized address value.

addrvar
is the name of the REXX variable to receive the resulting value. The value is returned as a character string and is in the form of the data in the dump.

PFXChar
indicates that a variable name follows as the next parameter. The variable named will receive the single-character prefix character which describes how the address was translated (L, R, or U). This value is derived from the prefix character specified on this subcommand, or the default set by the VMDTSET ADDRESS subcommand. PFXChar is allowed only for a CP dump.

ASIBK
returns all or a portion of the ASIBK record from the dump. The ASIBK record is mapped by the ASIBK control block (32-bit dump) or the ASIZBK control block (64-bit dump).

LENgth
specifies that the length value follows.

length
specifies the one- to four-digit decimal length of the data to be returned. The maximum length is 4096.

asivar
is the name of the REXX variable to receive the resulting value. The value is returned as a character string and is in the form of the data in the dump.

DFIR
returns all or a portion of the DFIR.

DFIZ
returns all or a portion of the DFIZ.

LENgth
specifies that the length value follows.

length
specifies the one- to five-digit decimal length of data to be returned. The maximum value allowed is 36,864, the size of a fully populated DFIZ. (See Usage Note 3).

dfvar
is the name of the REXX variable to receive the resulting value. The value is returned as a character string and is mapped by the DFIZ control block. (See Usage Note 3.)

DISPL
returns the four-character hexadecimal displacement of a selected field from the PFXPG or SYSCM control block.

FRom
FOr
indicates that the input address follows as the next parameter.

fieldname
specifies the name of the field for which to return the displacement.

TO
AS indicates that a variable name follows as the next parameter. The variable named will receive the standardized address value.

displvar
is the name of the REXX variable to receive the resulting value. The value is returned as readable hex characters.

FRMTE
locates the frame table entry (FRMTE) for a specified frame and returns the whole entry and/or specified fields in variables.

FRom
FOr
indicates that a page address follows as the next parameter.

frameaddr
specifies the one- to sixteen-significant digit hexadecimal real address of the frame for which the frame table entry is to be retrieved.

TO
AS indicates that a variable name follows as the next parameter.

frmtevar
is the name of the REXX variable to receive the contents of the entire frame table entry. The data is returned as a 64-digit readable hexadecimal value. The variable name may not be ADDRESS or any other keyword associated with this subcommand (or any abbreviations thereof).

ADDRess
specifies that the sixteen–digit hexadecimal logical address of the frame table entry is also to be returned.

addrvar
is the name of the REXX variable to receive the address of the frame table entry. The data is returned as a sixteen-digit readable hexadecimal value.

FRMCSB0
specifies that the contents of the one–byte FRMCSB0 field from the frame table entry are to be returned.

frmcsb0var
is the name of the REXX variable to receive the contents of the FRMCSB0 field from the frame table entry. The data is returned as a two–digit readable hexadecimal value.

FRMCSB1
specifies that the contents of the one–byte FRMCSB1 field from the frame table entry are to be returned.

frmcsb1var
is the name of the REXX variable to receive the contents of the FRMCSB1 field from the frame table entry. The data is returned as a two–digit readable hexadecimal value.

FRMCSB2
specifies that the contents of the one–byte FRMCSB2 field from the frame table entry are to be returned.

frmcsb2var
is the name of the REXX variable to receive the contents of the FRMCSB2 field from the frame table entry. The data is returned as a two–digit readable hexadecimal value.

FRMCSB3
specifies that the contents of the one–byte FRMCSB3 field from the frame table entry are to be returned.

frmcsb3var
is the name of the REXX variable to receive the contents of the FRMCSB3 field from the frame table entry. The data is returned as a two–digit readable hexadecimal value.

FRMID
specifies that the contents of the four–byte FRMID field from the frame table entry are to be returned.

frmidvar
is the name of the REXX variable to receive the contents of the FRMID field from the frame table entry. The data is returned as a four–character string value. Unreadable characters are translated to periods.

FRMFPNT
specifies that the contents of the eight–byte FRMFPNT field from the frame table entry are to be returned.

frmfpntvar
is the name of the REXX variable to receive the contents of the FRMFPNT field from the frame table entry. The data is returned as a 16–digit readable hexadecimal value.

FRMBPNT
specifies that the contents of the eight–byte FRMBPNT field from the frame table entry are to be returned.

frmbpntvar
is the name of the REXX variable to receive the contents of the FRMBPNT field from the frame table entry. The data is returned as a 16–digit readable hexadecimal value.

FRMPTE
specifies that the contents of the eight–byte FRMPTE field from the frame table entry are to be returned.

frmptevar
is the name of the REXX variable to receive the contents of the FRMPTE field from the frame table entry. The data is returned as a 16–digit readable hexadecimal value.

HEX31

HEX32

HEX64
specifies that an input value is to be normalized and checked for being a valid hexadecimal value which fits within the number of bits indicated. HEX31 and HEX32 allow up to eight characters; HEX64 allows up to seventeen characters (eight digits, an underscore character, and eight more digits).

FRom
FOr
indicates that a hexadecimal value follows as the next parameter.

value
specifies the one- to eight-significant digit (for HEX31 or HEX32) or the one- to sixteen-significant digit (for HEX64) hexadecimal value to be checked. The underscore character is ignored. The data is returned as a readable hexadecimal value, eight digits for HEX31 or HEX32 and sixteen digits for HEX64, is padded on the left with zeros, and does not include any underscores.

hexvar
is the name of the REXX variable to receive the resulting value.

MAPA
allows a macro to resolve an address to a module name.

MAPN
allows a macro to find the address of a module or entry point name.

FRom
FOr
indicates that a hexadecimal value follows as the next parameter.

addr
specifies the one- to eight- significant digit hexadecimal logical address to be mapped.

name
specifies the one- to eight- character name of the entry point or module to be looked up.

var
specifies the name of the variable to receive the result.

For the MAPA option, the result is returned as a fifteen-character field in one of the following formats (with a dot denoting a blank):

HCPxxx.....
the CP assembler module name.

HCPxxxEP.....
the CP assembler entry point name.

HCPxxx+dddd..
the displacement in a CP assembler module.

_xxxEP.....
the CP PLX module name.

_xxxEP+dddd....
the displacement in the CP PLX module.

xxxxxxxx.....
the non-CP module or entry point name.

xxxxxxxx+dddddd
the displacement in a non-CP PLX module.

For the MAPN option, the result is returned as a sixteen–digit right-aligned hexadecimal value that does not contain an underscore character.

SYMPTOM
returns all or a portion of the symptom record. The symptom record is mapped by the ADSR control block.

LENgth
specifies that the length value follows.

length
specifies the one- to four-digit decimal length of data to be returned. The maximum length is 4096.

symrecvar
is the name of the REXX variable to receive the resulting value. The value is returned as a character string and is in the format of the data in the dump.

Usage Notes

1. For all applicable EXTRACT options, the *var* parameter can precede the *addr* parameter if the associated TO, AS, or FOR keywords are included.
2. Frame table entries are returned in the following format:

32-bit dumps

0	00 00 00 00	word 0	00 00 00 00	word 1
10	00 00 00 00	word 2	00 00 00 00	word 3

64-bit dumps

double-word 0	double-word 1
double-word 2	double-word 3

In the dump from a 32-bit system (from a previous release), the high order half of each double-word is zero.

3. The DFIR in a 32-bit format dump is converted to the 64-bit DFIZ format with the following exceptions:
   - Within the 128-byte space for the registers, the general registers are four bytes long in the first 64 bytes.
   - DFIZFMT contains 00, the indicator of a 32-bit format dump.
   - The control registers are four bytes long.
   - There are only four 32-bit floating point registers.
   - The virtual PSW is eight bytes long.
4. The high order bit of the HEX31 operand is set to zero before the result is returned.
5. Usage notes for DISPL:
   - BLOCK with the SETVAR option is available for fields not supported by EXTRACT DISPL.
   - The displacement or equate value information returned is obtained directly from the dump for a version 5, release 1.0 or later CP dump, and from values stored in the VM Dump Tool for dumps associated with prior releases.
   - Only the following field names from the PFXPG and SYSCM are supported.

     ```
     PFXBALSV  PFXGEXTN  PFXILPSW  PFXSSABK  PFXSYSVM
     PFXCPIMG  PFXGEXTO  PFXIONP   PFXSTATE  PFXTMPSV
     PFXCPME   PFXGION   PFXIOOP   PFXSTBLG  PFXTYPE
     PFXCPRQA  PFXGIOO   PFXIRPSV  PFXSTBLG  PFXUDED
     PFXCPUID  PFXGMCHN  PFXLRQ    PFXSTBLG  PFXWRKSV
     PFXCRLG   PFXGMCHO  PFXMCHNP  PFXSTLEN  SYSEXITS
     PFXEXTNP  PFXGPRGN  PFXMCHOP  PFXSTLEN  SYSRIOIX
     PFXEXTOP  PFXGPRGO  PFXPRFIX  PFXSTPFX  SYSTODST
     PFXFEIBM  PFXGRSTN  PFXPRGNP  PFXSVCNP  SYSVOLCT
     PFXFRESV  PFXGRSTO  PFXPRGOP  PFXSVCOP  SYSVOLS
     PFXFTBLG  PFXGSVCN  PFXPTRSV  PFXSXASC
     PFXFTLEN  PFXGSVCO  PFXRSTOP  PFXSYS
     ```

   - This option is supported only for CP dumps.
6. Usage notes for MAPA/MAPN:
   - If syntax errors are encountered or if an address can not be resolved to any module, an error message is queued to the program stack, the variable is not set and a non-zero return code is returned.
   - For MAPA, the resulting value is based on the address as found in the associated VMDTMAP file.
   - For MAPN, the input address must match the content of the VMDTMAP file.
   - A few bytes of storage are often skipped between modules to force the latter module to the right address boundary. These addresses are reported as “not in a module.”
7. Usage note for FRMTE: this option is supported only for CP dumps.
8. Use VMDTSET DEBUG ON to have non-zero return codes and error messages displayed to the virtual machine console. For more information, see “About SET DEBUG” on page 203.
9. Multiple field names can be specified with the FRMTE function.

Examples

The following are examples of the EXTRACT subcommand. Note that the EXTRACT subcommand (in bold) can be issued only from a macro.

```
>>> extract address for L00EA0000 to sampout pfxchar to xxx
(RC is 0, sampout set to 000000000F270000, xxx set to L)

>>> extract address for R00EA0000 to sampout pfxchar to xxx
(RC is 0, sampout set to 0000000000EA0000, xxx set to R)

>>> extract frmte 0 frmcsb0 sampout
(RC= 0, sampout set to 01)

>>>extract frmte 0 address sampout
(RC= 0, sampout set to 00000000FE000000)

>>>extract hex31 f00 as sampout
(RC= 0, sampout set to 00000F00)

>>>extract hex31 ffffffff as sampout
(RC= 0, sampout set to 7FFFFFFF)

>>>extract hex31 00000000_ffffffff as sampout
(RC= 0, sampout set to 7FFFFFFF)

>>>extract hex31 123456789f as sampout
HCQEXT055E 123456789F too large, 00000000_7FFFFFFF max
(RC= 55, sampout is unchanged, contains 7FFFFFFF)

>>>extract hex31 abcdefg as sampout
HCQEXT003E ABCDEFG contains non-hex data
(RC= 3, sampout is unchanged, contains 7FFFFFFF)

>>>extract hex32 ffffffff as sampout
(RC= 0, sampout set to FFFFFFFF)

>>>extract hex64 123456789f as sampout
(RC= 0, sampout set to 000000123456789F)

>>>extract hex64 12345_6789f as sampout
(RC= 0, sampout set to 000000123456789F)

>>>extract hex64 !@#$%¬&*() as sampout
HCQEXT003E !@#$%¬&*() contains non-hex data
(RC= 3, sampout is unchanged, contains 000000123456789F)

>>>extract dfir 8 sampout
(RC= 0, sampout set to HCPDFIZ )

>>>extract symptom 8 sampout
(RC= 0, sampout set to SR206401)
```

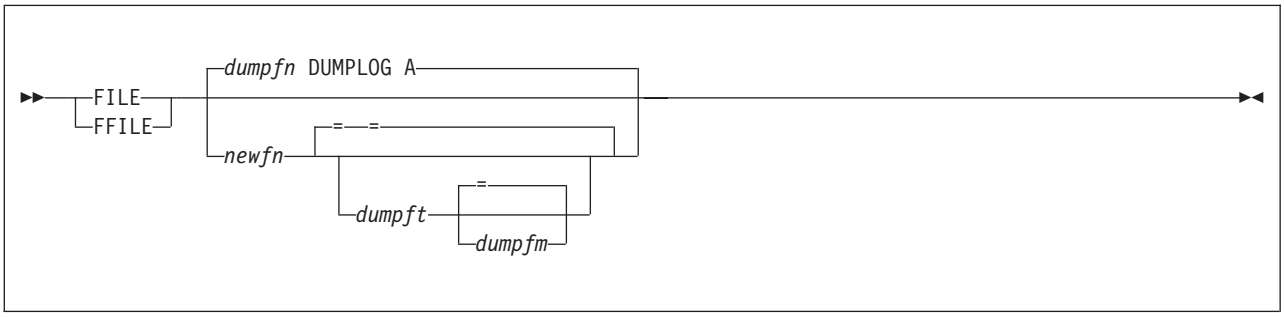
The following sample macro can be used to invoke the EXTRACT MAPA and EXTRACT MAPN subcommands:

```
/*--------------------------------------------------------------------*/
/* DOCMAP VMDT - example of EXTRACT MAPA, MAPN function               */
/*                                                                    */
/*--------------------------------------------------------------------*/
Address VMDUMPTL

/*--------------------------------------------------------------------*/
/* 'extract from name' the address of a module                        */
/*--------------------------------------------------------------------*/
cmd = 'EXTRACT MAPN FOR HCPSVC TO SVC@'   /* command to be issued */
Queue 'source command:' cmd               /* echo to dump session */
cmd                                       /* issue the command */
Queue 'return code >'rc'<   address returned is >'svc@'<'

Queue ' '                          /* blank line */
```

```
/*--------------------------------------------------------------------*/
/* convert that address back to a module name                         */
/*--------------------------------------------------------------------*/
cmd = 'EXTRACT MAPA' svc@ 'TO MODNAME'    /* command to be issued */
Queue 'source command:' cmd               /* echo to dump session */
cmd                                       /* issue the command */
Queue 'return code >'rc'<   name returned is >'modname'<'

Exit 0

The following was added to the dump session by invoking the above macro:

>>> docmap
source command: EXTRACT MAPN FOR HCPSVC TO SVC@
return code >0<   address returned is >00000000001D4530<

source command: EXTRACT MAPA 00000000001D4530 TO MODNAME
return code >0<   name returned is >HCPSVC     <
```

```
PI end
```

FILE/FFILE Subcommand



Purpose

The FILE subcommand saves the contents of the current session in the dump log file and terminates the VM Dump Tool session. The FFILE subcommand will override an existing dump log file if needed, when saving the contents of the current session.

Operands

dumpfn
is the file name of the dump itself.

newfn
is the new file name to be assigned to the dumplog before filing it.

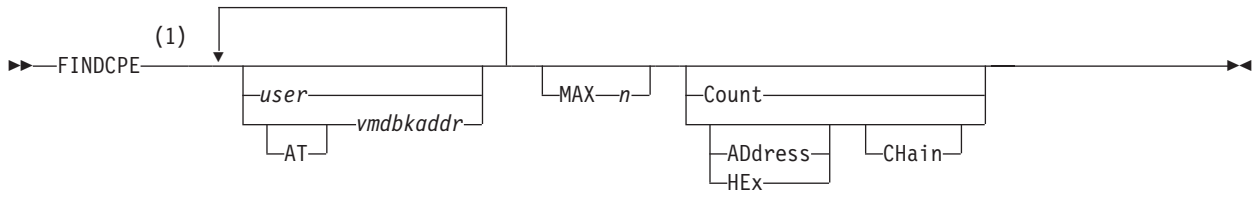
dumpft
is the new file type to be assigned to the dumplog before filing it.

dumpfm
is the new file mode to be assigned to the dump before filing it.

Usage Notes

1. Without other parameters, the dump log file will be saved as "*dumpname* DUMPLOG A", where *dumpname* is the file name of the dump file. The file name of the dump is displayed at the top of the XEDIT session screen, and can be obtained by the DUMPNAME subcommand.
2. See also the "SAVE/SSAVE Subcommand" on page 127 and the "QUIT/QQUIT Subcommand" on page 115 for related information.

FINDCPE Macro



Notes:

1 The default is all active CPEBKs. CAUTION: this could be a very large amount of data.

Purpose

The FINDCPE macro displays the contents of all active CPEBKs, SAVBKs, and SVGBKs (referred to generically below as 'CPEBKs'). for one or more specified users, or for all users in the system.

FINDCPE finds all CPEBKs for the specified user(s) that are either active (used for a call, but control has not returned) or have been obtained with HCPGETST but not yet released. This macro is very useful in diagnosing "hung user" problems, because it finds outstanding work for a user.

Operands

user
is the address of the VMDBK or the user ID of a user whose CPEBKs you want to find. One or more such addresses may be specified, separated by blanks.

AT *vmdbkaddr*
specifies VMDBK is the one- to eight-significant digit hexadecimal logical VMDBK address that will be used and will not be considered as a user ID.

MAX *n*
is the maximum number of CPEBKs to display. This parameter can be used to prevent running out of storage if there are too many CPEBKs.

Count
causes only a count of the number of active CPEBKs to be shown.

CHain
causes all CPEBKs found that meet the specified VMDBK criteria, to be examined for calling sequences (via CPEXR13), and sorted into chains according to these sequences.

HEx
indicates that each CPEBK should be displayed in hexadecimal instead of being formatted.

PI ADdress
indicates that only the address of each CPEBK should be displayed instead of the contents. **PI end**

Usage Notes

1. If no parameter is specified, all active CPEBKs in the system will be shown. Note that this option may produce such a large amount of output, and that your virtual machine storage could be exhausted. If in doubt, use the Count option.
2. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
3. Hex/Address output can be used as input to macros to do other processing.
4. If the CPEBK will be formatted, it is done by the CPEBK subcommand. The MAP/NOMAP and BITS/NOBITS options of the SET subcommand are used in displaying the CPEBKs.
5. FINDCPE handles CPEBKs, SAVBKs, and SVGBKs, generically referred to as CPEBKs.
6. This macro is supported only for CP dumps.
7. PLX work areas are recognized and interpreted, but the PLX work areas themselves are not displayed.

Examples

Typical use of and output from the FINDCPE macro:

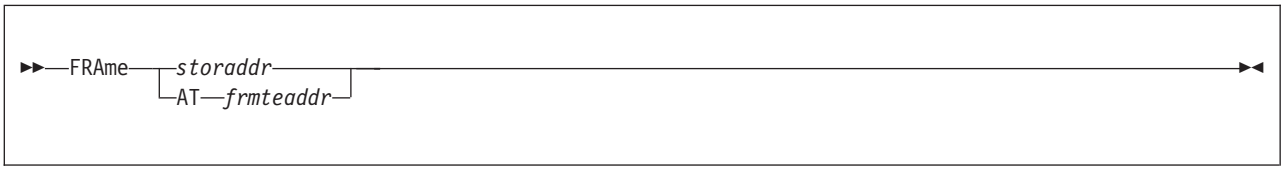
```
>>> findcpe operator
OPERATOR VMDBK at 770B4000
---- CPEBK at 773D0300
CPEXFPNT 00000000 CPEXBPNT 046B96F0 CPEXSFQP 00000000 CPEXCPRQ 046583A8
CPEXSCHC 00 CPEXCALC 80 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (00)
  Bits  defined in CPEXCALC         (80)
    80 Savearea in use for a call
R0-7 0000001F 00000000 00004010 00000000 0000001F 00000000 76D43508 770B4000
R8-F 7752FDF8 846624C2 770B4000 770B4000 04662270 04240500 8466250C 04746708
CPEXR14 is HCPRVC+29C
CPEXR15 is HCPQCNWR
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000050 *...................&*
CPEXWRK5-9 770FDD46 00000000 00000000 00000000 00000000 *....................*
---- CPEBK at 04240500
CPEXFPNT 00000000 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 01 CPEXCALC 80 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (01)
    01 Dispatch on the master CPU only
  Bits  defined in CPEXCALC         (80)
    80 Savearea in use for a call
R0-7 0000001F 7751DD08 00004010 00000000 0000001F 00000000 77631DA0 77631D60
R8-F 7752FDF8 7751DCF8 04100458 770B4000 0476BB90 0431ED80 8476C2DE 04662350
CPEXR14 is HCPVCO+74E
CPEXR15 is HCPRVCWT
CPEXWRK0-4 00000000 00000000 00004010 00000000 00000000 *....................*
CPEXWRK5-9 00000000 00000000 0000001F 7751DD08 00000000 *....................*
---- CPEBK at 0431ED80
CPEXFPNT 00000000 CPEXBPNT 046B96F0 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 00 CPEXCALC 80 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (00)
  Bits  defined in CPEXCALC         (80)
    80 Savearea in use for a call
R0-7 0000001F 7751DD08 00004010 00000000 7711CC80 00000000 77631DA0 77631D60
R8-F 7752FDF8 7751DCF8 04100458 770B4000 0476AF80 00000000 8476B88A 0476C288
CPEXR14 is HCPVCN+90A
```

```
CPEXR15 is HCPVCOWR
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *.....................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *.....................*
3 active CPEBKs were processed
```

>>> findcpe operator chain

```
OPERATOR VMDBK at 770B4000

*** Start of CPEBK chain ***

---- CPEBK at 773D0300
CPEXFPNT 00000000 CPEXBPNT 046B96F0 CPEXSFQP 00000000 CPEXCPRQ 046583A8
CPEXSCHC 00 CPEXCALC 80 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (00)
  Bits  defined in CPEXCALC         (80)
    80 Savearea in use for a call
R0-7 0000001F 00000000 00004010 00000000 0000001F 00000000 76D43508 770B4000
R8-F 7752FDF8 846624C2 770B4000 770B4000 04662270 04240500 8466250C 04746708
CPEXR14 is HCPRVC+29C
CPEXR15 is HCPQCNWR
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000050 *...................&*
CPEXWRK5-9 770FDD46 00000000 00000000 00000000 00000000 *.....................*
---- CPEBK at 04240500
CPEXFPNT 00000000 CPEXBPNT 00000000 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 01 CPEXCALC 80 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (01)
    01 Dispatch on the master CPU only
  Bits  defined in CPEXCALC         (80)
    80 Savearea in use for a call
R0-7 0000001F 7751DD08 00004010 00000000 0000001F 00000000 77631DA0 77631D60
R8-F 7752FDF8 7751DCF8 04100458 770B4000 0476BB90 0431ED80 8476C2DE 04662350
CPEXR14 is HCPVCO+74E
CPEXR15 is HCPRVCWT
CPEXWRK0-4 00000000 00000000 00004010 00000000 00000000 *.....................*
CPEXWRK5-9 00000000 00000000 0000001F 7751DD08 00000000 *.....................*
---- CPEBK at 0431ED80
CPEXFPNT 00000000 CPEXBPNT 046B96F0 CPEXSFQP 00000000 CPEXCPRQ 04520000
CPEXSCHC 00 CPEXCALC 80 CPEXFORM 00 CPEXRETN 0467D040 (32-bit)
  Bits  defined in CPEXSCHC         (00)
  Bits  defined in CPEXCALC         (80)
    80 Savearea in use for a call
R0-7 0000001F 7751DD08 00004010 00000000 7711CC80 00000000 77631DA0 77631D60
R8-F 7752FDF8 7751DCF8 04100458 770B4000 0476AF80 00000000 8476B88A 0476C288
CPEXR14 is HCPVCN+90A
CPEXR15 is HCPVCOWR
CPEXWRK0-4 00000000 00000000 00000000 00000000 00000000 *.....................*
CPEXWRK5-9 00000000 00000000 00000000 00000000 00000000 *.....................*
3 active CPEBKs were processed
```

FRAME Subcommand



Purpose

The FRAME subcommand displays information about a frame of storage from its frame table entry. This is separate and distinct from the FRAMES subcommand.

Operands

storaddr
is the one- to sixteen-significant digit hexadecimal real address in the dump for which the frame table entry is to be displayed.

AT *frmteaddr*
is the one- to sixteen-significant digit hexadecimal logical address in the dump of the frame table entry to be displayed.

Usage Notes

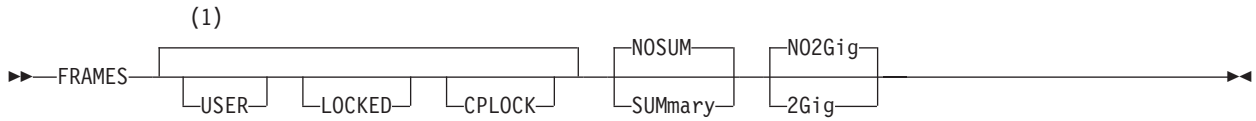
1. You may specify *storaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. If you issue VMDTSET BITS ON has been issued, bit definitions for the FRMCSBx bytes are also displayed.
3. This subcommand is supported only for CP dumps.

Examples

Typical use of and output from the FRAME subcommand:

```
>>> frame 00EA0000
Frame table for page 00000000_00EA0000 is at 00000000_8001D400
FRMFPNTG  00000000_80164140    *....... *
FRMBPNTG  00000000_801D2C60    *.......-*
FRMPTEG   00000000_09C0CC60    *.....{.-*
FRMSTATEG 00000000_80200000    *........*
FRMCSB0 80 FRMCSB1 20 FRMCSB2 00 FRMCSB3 00

>>> dts bits on
complete

>>> frame 00EA0000
Frame table for page 00000000_00EA0000 is at 00000000_8001D400
FRMFPNTG  00000000_80164140    *....... *
FRMBPNTG  00000000_801D2C60    *.......-*
FRMPTEG   00000000_09C0CC60    *.....{.-*
FRMSTATEG 00000000_80200000    *........*
FRMCSB0 80 FRMCSB1 20 FRMCSB2 00 FRMCSB3 00
  Codes defined in FRMCSB0          (80)
    80 FRMUSER FRAME USED AS USER PAGE
  Bits  defined in FRMCSB1          (20)
    20 FRMOWNED Frame is on a user owned list
  Bits  defined in FRMCSB2          (00)
  Bits  defined in FRMCSB3          (00)
```

FRAMES Subcommand



Notes:

1 The default is a summary of all frames in the frame table.

Purpose

The FRAMES subcommand displays statistical information from the frame table. If no options are specified, a summary is given for all the frames in the frame table.

Operands

USER
requests a list of user pages. Note that there may be a large number of pages, and it may take a long time to get through them.

LOCKED
causes only locked frames to be listed. Used in conjunction with the USER option, this will locate all locked user pages.

CPLOCK
causes only frames locked with the CP LOCK command to be listed.

NOSUM
indicates that all lines should be displayed even if the count is zero.

SUMmary
indicates that lines with zero counts should not be displayed.

NO2Gig
indicates that all of real storage should be analyzed.

2Gig
indicates that only real storage below the 2 GB line should be analyzed.

Usage Notes

1. To halt a long-running FRAMES command, press the PA1 key to break out of full screen mode and then enter the HI command.
2. This subcommand is supported only for CP dumps.
3. The count of available pages below and above the 2 Gig line are reported separately.

Examples

Typical use of and output from the FRAMES subcommand:

```
>>> frames
         0 Unknown frames
         0 offline frames
         0 frames not initialized
```

```
  40960 Frame Table frames
   4096 SXS Table frames
   4392 free storage frames
           0 of which are CPEBK frames
        4177 of which are SHORTTERM frames
           0 of which are GUESTPERM frames
           0 of which are SYSPERM frames
         215 of which are ALIGNED FREE frames
           0 of which are VERIFIABLE CB frames
           0 of which are Persistent SYSPERM frames
  14726 CP frames
    416 trace table frames
    130 prefix frames
   4679 pageable PGMBK frames
   4679 2nd pageable PGMBK frames
   2757 non-pageable PGMBK frames
   2757 2nd non-pageable PGMBK frames
 607568 user frames
           0 of which are locked
   1753 system virtual frames
   5144 system utility address space frames
 451544 frames marked available, < 2G
      451452 of which are actually available
           0 of which are on a local available list
          92 of which are in transit
4004361 frames marked available, >= 2G
     4003978 of which are actually available
           0 of which are on a local available list
         383 of which are in transit
   5288 Data Space frames
      0 VDISK frames
  87438 MDC Addr Space frames
    180 MDC Seg/Page frames
     12 MDC Hash Table frames
5242880 frames in total

>>> frames sum 2g

     106 free storage frames
          101 of which are SHORTTERM frames
            5 of which are ALIGNED FREE frames
   3768 CP frames
    400 trace table frames
    130 prefix frames
  62525 user frames
      2 system virtual frames
    181 system utility address space frames
 451544 frames marked available, < 2G
      451452 of which are actually available
          92 of which are in transit
   1389 Data Space frames
   4063 MDC Addr Space frames
    180 MDC Seg/Page frames
 524288 frames in total
```

FREGS Subcommand

Notes:

1 The default is the abending processor.

Purpose

The FREGS subcommand displays the floating point registers from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

Usage Notes

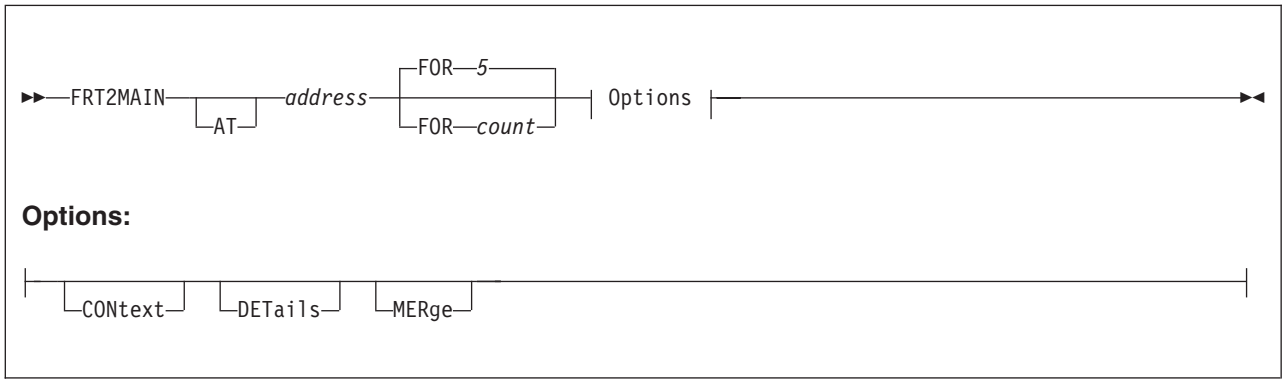
You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.

Examples

Typical use of and output from the FREGS subcommand:

```
>>> fregs
F0-6   0000000000000000  0000000000000000  0000000000000000  0000000000000000

>>> fregs
Floating Point Control Register 00000000
F0-3   0000000000000000  0000000000000000  0000000000000000  0000000000000000
F4-7   0000000000000000  0000000000000000  0000000000000000  0000000000000000
F8-B   0000000000000000  0000000000000000  0000000000000000  0000000000000000
FC-F   0000000000000000  0000000000000000  0000000000000000  0000000000000000
```

FRT2MAIN Macro



Purpose

The FRT2MAIN macro accepts the address of an entry in a Function Related trace table and returns the address of the corresponding entry in the main trace table.

Operands

address
The one- to eight-significant digit hexadecimal logical address of a trace entry in a function-related trace table for which a corresponding entry in the main trace table should be found.

AT indicates that the address follows. This provides consistency with other macros and commands that support AT.

CONtext
specifies that the resulting trace entry in the main trace table should be displayed with associated trace entries before and after it. Exactly what is displayed depends further on the values of the MERGE and FOR parameters.

DETails
specifies that a number of internal calculation results should be displayed in the output.

FOR *count*
allows the user to specify how many trace entries should be displayed before and after the trace entry of interest in output for the CONTEXT parameter. The FOR option also sets the CONTEXT option. The default for *count* is 5.

MERge
specifies that the output for the CONTEXT option should display trace information for all CPUs found in the dump. The default is to display trace information only for the failing CPU. The MERGE option also sets the CONTEXT option.

Usage Notes

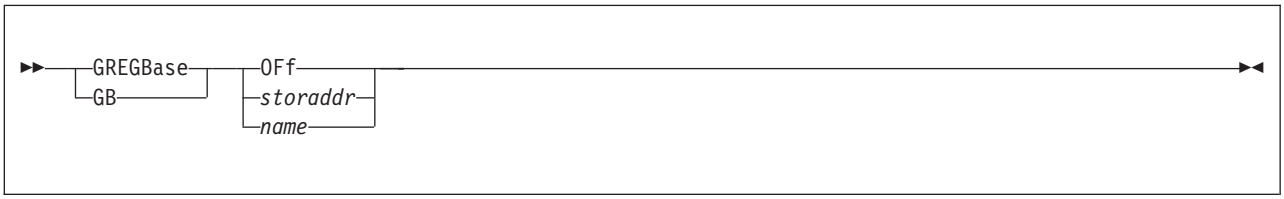
1. Note that Function Related trace entry addresses are logical addresses, not real addresses.
2. The MERGE, CONTEXT and DETAILS options may be specified in any order.

Examples

Typical use of and output from the FRT2MAIN macro:

```
>>> frt2main 0148A040
Fcn-Related Trace at 0148A040 ---> main Trace at 00000000_079E06C0
079E06C0 07:09:33  0000 AAAAAAAA_00000003 AAAAAAAA_00000008
>>> frt2main 0148A040 context
Fcn-Related Trace at 0148A040 ---> main Trace at 00000000_079E06C0
079E07E0 07:11:45  Call fr HCPFRT+248 to HCPSXPDF cpebk 00B5F200
079E07A0 07:11:45  Monitor Call at HCPSVC+56C8
079E0780 07:09:43  Call fr HCPDST+22C to HCPFRTOP cpebk 00B5FE00
079E0740 07:09:43  Monitor Call at HCPSVC+3006
079E0700 07:09:33  E000 AAAAAAAA_00000003 AAAAAAAA_00000008
--------
079E06C0 07:09:33  0000 AAAAAAAA_00000003 AAAAAAAA_00000008
--------
079E0680 07:09:33  0000 AAAAAAAA_00000003 AAAAAAAA_00000008
079E0660 07:09:33  Rtrn to HCPDST+22C fr HCPFRT+332 cpebk 00B2F000
079E0640 07:09:33  Rtrn to HCPFRT+2EE fr HCPSXP+B62 cpebk 00B60600
079E0600 07:09:33  Monitor Call at HCPSXP+4654
>>> frt2main 0148A080 details
input_address 0148A080
trace_entry_format 7580   frame_type 21   trace_entry_code E000
cpu 0000   pfxpg 00032000   current_trace_page 00BA5000
page_entry_time: C3B4F456B67E8320 02/06/09 07:16:13.649896
page_exit_time : C3B4F2A2650EE568 02/06/09 07:08:36.137198
search_for_time: C3B4F2D8D0726D60 02/06/09 07:09:33.200167
in decimal     : 14102163346227883360 (20 digits)
Not in current trace page at 00BA5000
TOD is within the range of page 00C2E000
searching page 00C2E000
Fcn-Related Trace at 0148A080 ---> main Trace at 00000000_079E0700
079E0700 07:09:33  E000 AAAAAAAA_00000003 AAAAAAAA_00000008
```

GREGBASE Subcommand



Purpose

The GREGBASE subcommand sets the base address that will be used for all references to the abending processor's general registers.

Operands

OFf
specifies that the registers are from the time of the abend.

storaddr
is the one- to eight-significant digit hexadecimal logical address in the dump upon which to base the registers. This could be useful when the registers are in a dynamic savearea, but remember that the address you specify here should be the address where the registers are saved, and *not* the address of the savearea itself.

name
specifies that the registers be taken from one of the prefix saveareas. The name may be FREEsave or PFXFresv, BALRsave or PFXBalsv, TEMPsave or PFXTmpsv, PTRSave or PFXPtrsv, WORKsave or PFXWrksv.

Usage Notes

You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.

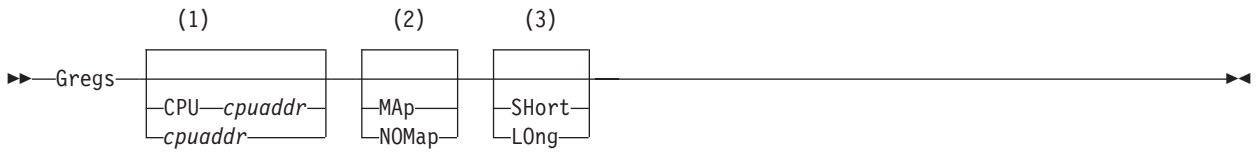
Examples

Typical use of and output from the GREGBASE subcommand and the interaction between the GREGS and GREGBASE subcommands follow:

```
>>> gb 720
General registers from 00000720
R0 00040011_00800000 HCWAI3+4170     R8 AAAAAAAA_0030D580 HCPNSUDA
R1 00000000_00001180                 R9 AAAAAAAA_80B4EA4E HCPUSO+7EE
R2 00000000_00000004                 RA AAAAAAAA_00000000
R3 00000000_0021A4D2 HCPHTT+CA2      RB 00000000_0CCB3000
R4 00000000_08889001                 RC 00000000_0030A900 HCPNSP
R5 00000000_00000001                 RD 00000000_03C3BE00
R6 00000000_00202F28 HCPHPC+8D8      RE 00000000_8030A9F6 HCPNSP+F6
R7 AAAAAAAA_042F5000                 RF 00000000_801DEE00 HCPFRE

>>> greg
General registers from 00000720
R0 00040011_00800000 HCWAI3+4170     R8 AAAAAAAA_0030D580 HCPNSUDA
R1 00000000_00001180                 R9 AAAAAAAA_80B4EA4E HCPUSO+7EE
R2 00000000_00000004                 RA AAAAAAAA_00000000
```

```
R3 00000000_0021A4D2 HCPHTT+CA2      RB 00000000_0CCB3000
R4 00000000_08889001                 RC 00000000_0030A900 HCPNSP
R5 00000000_00000001                 RD 00000000_03C3BE00
R6 00000000_00202F28 HCPHPC+8D8      RE 00000000_8030A9F6 HCPNSP+F6
R7 AAAAAAAA_042F5000                 RF 00000000_801DEE00 HCPFRET
```

GREGS Subcommand



Notes:

1 The default is the abending processor.

2 The default is the setting of VMDTSET REGS.

3 The default is based on the dump type; for 32–bit dumps, the default is SHORT, and for 64–bit dumps, the default is LONG.

Purpose

The GREGS subcommand displays the general registers from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is the one-to four-digit hexadecimal CPU address. The default is the abending processor, unless a general register base has been set previously with the GREGBASE subcommand.

MAP
the value of each register is mapped to a module and displacement.

NOMAP
Register values are not to be mapped to modules.

SHORT
Register values are assumed to be four bytes long.

LONG
Register values are assumed to be eight bytes long.

Usage Notes

1. The output format of this subcommand depends upon the size of the registers in the dump and the options established by the SET subcommand.
2. You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
3. The options apply only to this subcommand and override the options established by the SET subcommand.
4. If a GREGBASE subcommand was issued previously, the registers displayed with GREGS will be from the area specified by the GREGBASE with an indication of that. To reset this so the registers at the time of the abend are displayed, issue GREGBASE OFF.

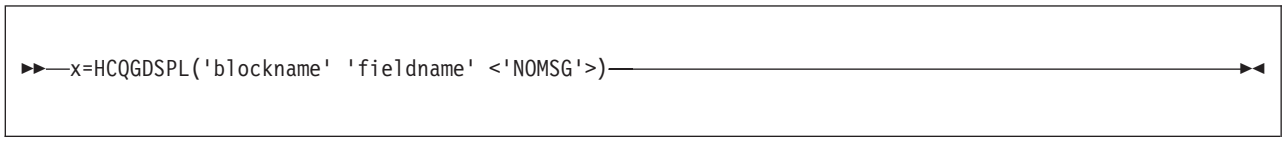
Examples

Typical output from the GREGS subcommand:

```
>>> greg
R0 00000000_00000000                R8 00000000_8002FC60
R1 00000000_01D16820                R9 00000000_8002FC60
R2 00000200_0003DAA0 HCPOPR+1BB0    RA 00000000_002B0830 HCPRSMCM
R3 00000000_002B0A98 HCPRSMAQ       RB 00000000_01DED000
R4 00000000_8002C080 HCPWRK+19080   RC 00000000_00062520 HCPALF
R5 00000000_01D16000                RD 00000000_0177B600
R6 00000000_000633E0 HCPALF+EC0     RE 00000000_09CB8F28
R7 00000000_00185210 HCPHTU+5C0     RF 00000000_80200010 HCPMCS+4B0

>>> greg 2
Data for CPU 0002
R0 00000003_00000000                R8 00000000_0000E000 HCPSYSLC
R1 FFFFFFFF_20000000                R9 00000000_802C7142 HCPSGP+972
R2 00000000_00000000                RA 00000000_00003000 HCPSYSMP
R3 00000000_00000000                RB 00000000_00002000 HCPSYSVM
R4 00000000_00000000                RC 00000000_002C67D0 HCPSGP
R5 00000000_00125FEC HCPDSP+12EC    RD 00000000_00000018
R6 00000000_80125B18 HCPDSP+E18     RE 00000000_801458CC HCPEXT+26C
R7 00000000_00D639F0                RF 00000000_00001000

>>> gb 720
General registers from 00000720
R0 00000000_00000000                R8 00000000_090A6800
R1 00000000_000000C0                R9 00000000_090A6800
R2 00000000_00000000                RA 00000000_01DED000
R3 00000000_00000051                RB 00000000_01DED000
R4 00000000_00000000                RC 00000000_00274380 HCPPAG
R5 00000000_0000E000 HCPSYSLC       RD 00000000_00C65A00
R6 00000000_002B0830 HCPRSMCM       RE 00000000_80274642 HCPPAG+2C2
R7 00000000_00185210 HCPHTU+5C0     RF 00000000_00000000

>>> greg
General registers from 00000720
R0 00000000_00000000                R8 00000000_090A6800
R1 00000000_000000C0                R9 00000000_090A6800
R2 00000000_00000000                RA 00000000_01DED000
R3 00000000_00000051                RB 00000000_01DED000
R4 00000000_00000000                RC 00000000_00274380 HCPPAG
R5 00000000_0000E000 HCPSYSLC       RD 00000000_00C65A00
R6 00000000_002B0830 HCPRSMCM       RE 00000000_80274642 HCPPAG+2C2
R7 00000000_00185210 HCPHTU+5C0     RF 00000000_00000000
```

HCQGDSPL Function

PI



Purpose

Macros written for the VM Dump Tool often need to include displacements of given fields in a control block to be able to pick up desired data from those fields. The HCQGDSPL function provides a quick way for a macro to obtain this information from the BLOCK data for the release.

Operands

x The variable to receive the result. This can be any valid REXX variable name.

= The REXX syntax to assign a value to a variable. HCQGDSPL must be called as a function.

'blockname'
The name of the control block which contains the desired field.

'fieldname'
The field name within that control block for which the displacement is to be returned.

'NOMSG'
An optional operand that prevents the display of messages from HCQGDSPL. This is useful when the field name might not exist in the BLOCK data that is in use. The calling macro should use the return code returned by HCQGDSPL to ensure that it was successful before proceeding.

Usage Notes

1. While HCQGDSPL will operate correctly without the single quotation marks if values have not been assigned to the variable, the inclusion of the quotation marks is preferred.
2. Note that the syntax is a blank between the parameters, not a comma.
3. HCQGDSPL returns a 4-digit hexadecimal value of the displacement of the given field.
4. If HCQGDSPL detects any problems, it will display an appropriate message (unless the NOMSG operand was specified) and provide a return code of the negative value of that message number.

Examples

In this example, we'll use HCQGDSPL to get the displacement of the field VMDUSER in the VMDBK.

The program, PLAY VMDT:

```
/* PLAY VMDT - display userid from VMDBK address */
@VMDUSER = HCQGDSPL('vmdbk' 'vmduser')          /* get userid displ */
Queue '@VMDUSER was set to' @VMDUSER
```

```
Arg vmdbk@ .                                        /* get vmdbk@ */
'SETVAR string userid' XADD(vmdbk@ @VMDUSER) '8'   /* get userid */
Queue 'the userid for vmdbk at' vmdbk@ 'is' userid /* display result */
Exit
/*--------------------------------------------------------------------*/
/* XADD Subroutine - Add 2 hex numbers to a hex result                */
/*--------------------------------------------------------------------*/
XADD: Procedure                   /* Hex ADD */
   Parse Arg h1 h2 .              /* h1 is Hex, h2 is Hex */
   Return D2X((X2D(h1) + X2D(h2)), 8)
```

Here is the output:

```
>>> vmdbk operator
OPERATOR VMDBK at 01236000

>>> play 1236000
@VMDUSER was set to 0200
the userid for vmdbk at 1236000 is OPERATOR
```

PI end

HEX Subcommand

Purpose

The HEX subcommand does simple hexadecimal calculations and displays the results in both hexadecimal and decimal format.

Operands

expression
is the expression to be evaluated. No imbedded blanks are allowed.

Usage Notes

1. You may specify any single parameter as an underscore character ("_". Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.

   Typical operations you can perform are:

   ```
   +  Addition       -  Subtraction    /  Division    *  Multiplication
   R  Shift right    L  Shift left     N  And         O  Or
   X  Exclusive or
   ```

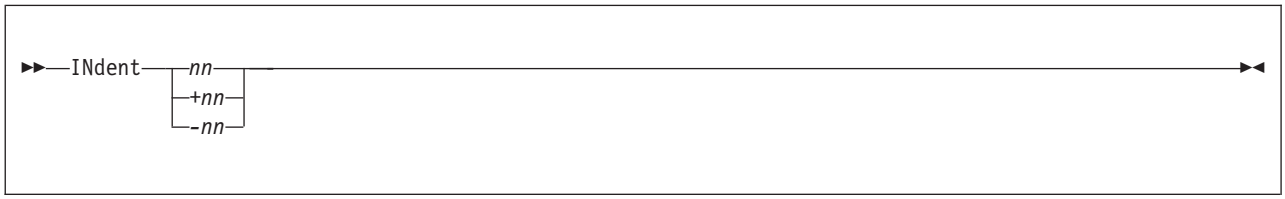
2. Note that the expression is evaluated strictly from left to right. No operation takes precedence over another.
3. The HEX subcommand supports only 31–bit numbers and arithmetic.

Examples

Typical use of and output from the HEX subcommand:

```
>>> hex 1000*4
Hex 00004000 Dec 16384

>>> hex 18/3
Hex 00000008 Dec 8
```

INDENT Subcommand



Purpose

The INDENT subcommand sets the offset from the left margin for output. It is usually used only within macros.

Operands

nn
: is a one- to two-digit decimal number that specifies an absolute indentation of *nn* spaces.

+nn
: specifies an increased indentation of *nn* spaces.

-nn
: specifies a decreased indentation of *nn* spaces.

Usage Notes

1. Note that this command is maintained for compatibility reasons only. The VMDTSET INDENT subcommand is the preferred method of invoking this function.

   See the "VMDTSET Subcommand" on page 171 for more information.
2. The current setting of this subcommand can be obtained with the VMDTQRY INDENT subcommand.

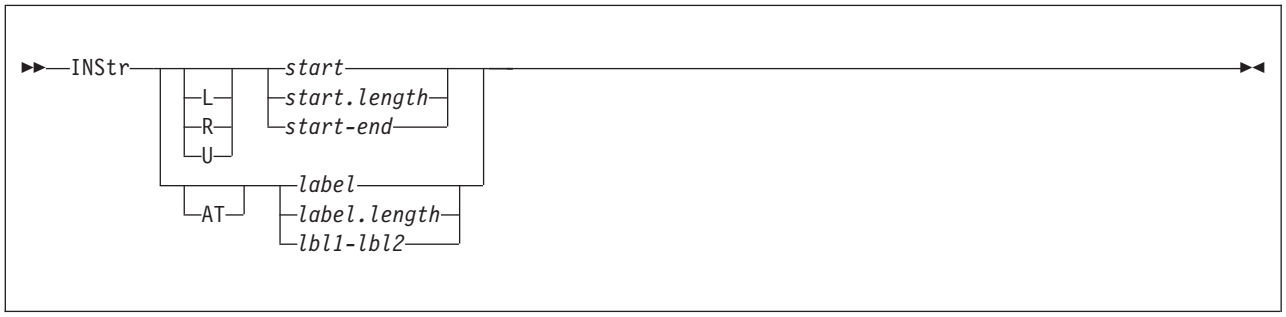
   See the "VMDTQRY Subcommand" on page 165 for more information.
3. If output shifted to the right gets too long for the output line, it will be truncated with a warning message.
4. The output of QUERY and VMDTQRY are not affected by the INDENT setting.

Examples

Typical use of and output from the INDENT subcommand:

```
>>> display 0.10
00000000_00000000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*

>>> indent +2

>>> display 0.10
  00000000_00000000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*
```

INSTR Subcommand



Purpose

The INSTR subcommand displays storage formatted as Assembler language instructions.

Operands

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *start*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

start
specifies the one- to eight-significant digit hexadecimal start address of the display.

length
specifies the one- to four-digit hexadecimal length of the display.

end
specifies the one- to eight-significant digit hexadecimal end address of the display. The end address must not be lower than the start address.

AT specified that a label follows.

label
specifies the module or entry point name where the display should begin.

lbl1
lbl2
specify the names of modules or entry points in the dump that define the limits of data to be displayed. The address of *lbl1* must be less than the address of *lbl2*. *lbl1* and *lbl2* are interpreted as logical addresses.

Usage Notes

1. *Start* and *end* may be specified as a hexadecimal number, a register in the form (Rx), a register plus hexadecimal offset in the form (Rx+ooo), a module or entry point name in the form xxxxxxxx, or a module or entry point name plus hexadecimal offset in the form xxxxxxxx+ooo.
2. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.

3. Strings of data that are not recognized as instructions are displayed in hexadecimal.
4. The name of the library currently in use is available via the VMDTQRY MNEMLIB subcommand. This list can be set or overridden by the VMDTSET MNEMLIB subcommand.

Examples

Typical use of and output from the INSTR subcommand:

```
>>> instr 1813E8.40
00000000_001813E8 +0000 A7F40017     BrU   X'17'
00000000_001813EC +0004 EB2906300024 STMG  R2,R9,X'630'
00000000_001813F2 +000A D20106AE08F2 MVC   X'6AE'(2),X'8F2'
00000000_001813F8 +0010 E3C008E80024 STG   R12,X'8E8'
00000000_001813FE +0016 E3C006A80017 LLGT  R12,X'6A8'
00000000_00181404 +001C 9B0FC000     STAM  R0,R15,0(R12)
00000000_00181408 +0020 E3C008E80004 LG    R12,X'8E8'
00000000_0018140E +0026 92E00613     MVI   X'613',X'E0'
00000000_00181412 +002A A7F40007     BrU   7
00000000_00181416 +002E EB2906300024 STMG  R2,R9,X'630'
00000000_0018141C +0034 92600613     MVI   X'613',X'60'
00000000_00181420 +0038 9A0A0A00     LAM   R0,R10,X'A00'
00000000_00181424 +003C E31010000004 LG    R1,0(,R1)

>>> instr at hcphtr7n.20
00000000_001813E8 +0000 A7F40017     BrU   X'17'
00000000_001813EC +0004 EB2906300024 STMG  R2,R9,X'630'
00000000_001813F2 +000A D20106AE08F2 MVC   X'6AE'(2),X'8F2'
00000000_001813F8 +0010 E3C008E80024 STG   R12,X'8E8'
00000000_001813FE +0016 E3C006A80017 LLGT  R12,X'6A8'
00000000_00181404 +001C 9B0FC000     STAM  R0,R15,0(R12)
00000000_00181408 +0020 E3C008E80004 LG    R12,X'8E8'
```

KEY Subcommand

Purpose

The KEY subcommand displays the storage key for the referenced page. If the page is a user page, an attempt is made to get the referenced and changed byte.

Operands

storaddr
: is the one- to sixteen-significant digit hexadecimal real address in the dump of the frame for which the key is to be obtained. The address does not have to be on a page boundary; the low order 12 bits are ignored.

Usage Notes

1. You may specify *storaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. The key value for a page is set in the dump only when that page is actually included in the dump. KEY displays the value found; it does not check the page status. Use the DISPLAY command for the same address to determine if the page is in the dump.

Examples

Typical use of and output from the KEY subcommand:

```
>>> key 10000
  Key for absolute address 00000000_00010000 is 06
```

LASTTRAN Macro

Purpose

The LASTTRAN macro can be used to display how the last logical or user-defined address was translated.

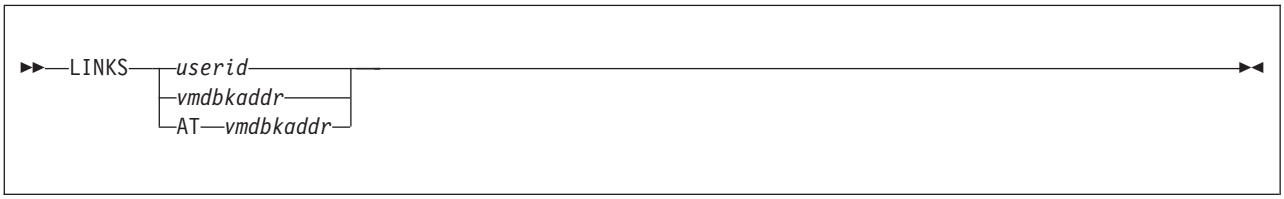
Usage Notes

This macro is supported only for CP dumps.

Examples

```
>>> display 017BB000.10
_017BB000 +0000 10010024 00000000 00000000 00000000 *.............*

>>> lasttran
Analysis of  00000000_017BB000 translated on ASCE 00000000_0000A007
Table type Table address     + displ  contains
---------- ----------------- -------- -----------------
Region 1st (not applicable)
Region 2nd (not applicable)
Region 3rd 00000000_0000A000 00000000 00000000_0FFF4007
Segment    00000000_0FFF4000 000000B8 00000000_0FB86800
Page       00000000_0FB86800 000005D8 00000000_0F01F000
           (translation was successful)

>>> lasttran details
Analysis of  00000000_017BB000 translated on ASCE 00000000_0000A007
Index: Region 1st  Region 2nd  Region 3rd     Segment     Page
bits  00000000000 00000000000 00000000000 00000010111 10111011
hex           000         000         000         017       BB
displ    00000000    00000000    00000000    000000B8 000005D8

Table type Table address     + displ  contains
---------- ----------------- -------- -----------------
Region 1st (not applicable)
Region 2nd (not applicable)
Region 3rd 00000000_0000A000 00000000 00000000_0FFF4007
Segment    00000000_0FFF4000 000000B8 00000000_0FB86800
Page       00000000_0FB86800 000005D8 00000000_0F01F000
           (translation was successful)

>>> display ffffffff.10
HCQNID004E Error encountered on address 00000000_0FFFBFF8
Error 05, Translation error, use LASTTRAN for details

>>> lasttran
Analysis of  00000000_FFFFFFFF translated on ASCE 00000000_0000A007
Table type Table address     + displ  contains
---------- ----------------- -------- -----------------
Region 1st (not applicable)
Region 2nd (not applicable)
Region 3rd 00000000_0000A000 00000008 00000000_0FFF8007
Segment    00000000_0FFF8000 00003FF8 00000000_00000020
           ---> Table entry is marked invalid
```

LINKS Macro



Purpose

The LINKS macro accepts an input of either a user ID or a VMDBK address, and displays one line for each LINK to a minidisk that is found for the user ID, along with relevant control block addresses.

Operands

userid
is the one- to eight-character user ID of the VMDBK to be analyzed.

vmdbkaddr
AT *vmdbkaddr*
is the one- to eight-significant digit hexadecimal logical address of the VMDBK to be analyzed.

Usage Notes

1. If AT is not specified, the input parameter is first tested as a VMDBK address. If that fails, then it is tested as a user ID.
2. The LINKS macro obtains VMDCHRDN from the VMDBK (pointer to the radix tree of devices by device number). For each non-zero entry in the radix tree, it examines the VDEV to see if it represents a LINK to another device, and (if it does) formats and prints the pertinent information and control block addresses.
3. There is no way to limit the output of this macro except to use the CMS HI function to halt interpretation.
4. You may specify *userid* or *vmdbkaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the string pointed to by the cursor.
5. This macro is supported only for CP dumps.

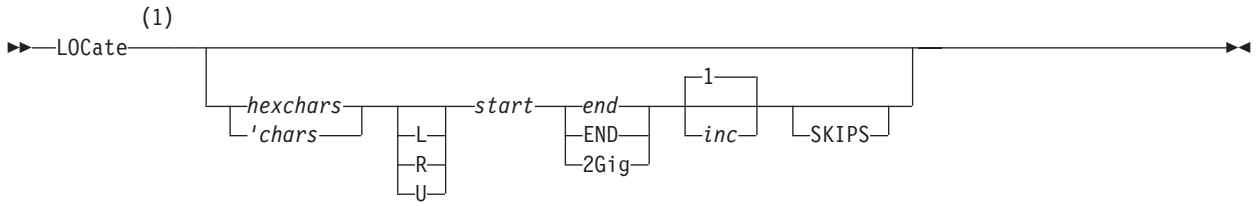
Examples

Typical use of and output from the LINKS macro:

```
>>> links operator
Minidisk LINKs for VMDBK 770B4000  User OPERATOR
         userid   vdev    vdev R/x vdev@    rdev rdev@    mdisk@
    LINK CMSDISK  0007 as 0007 R/O 76535EE8 E739 776089D8 76539268
    LINK SYSBUILD 0013 as 0013 R/O 76535A68 E408 775E6DF8 7653D368
    LINK SYSBUILD 0014 as 0014 R/O 76535828 E408 775E6DF8 7653D168
    LINK SYSBUILD 0015 as 0015 R/O 76535708 E400 775E5198 7653D0C0
    LINK SYSBUILD 0016 as 0016 R/O 765355E8 E41B 775E93A8 765793B0
    LINK OPERATOR 0191 as 0191 R/W 77631B78 E434 77522DA0 76538230
    LINK MAINT    019D as 019D R/O 776312C0 E408 775E6DF8 76539460
    LINK MAINT    019E as 019E R/O 776311A0 E400 775E5198 765393B8
    LINK MAINT    019F as 019F R/O 77631080 E408 775E6DF8 76539310
    LINK CMSDISK  0443 as 0443 R/O 76535B88 E432 775EBDF8 7653D468
```

```
    LINK SYSBUILD 0592 as 0592 R/O 76535948 E433 775EBBE8 7653D210
    LINK CMSDISK  0AAA as 0AAA R/O 76535CA8 E400 775E5198 76539010
    LINK CMSDISK  0BBB as 0BBB R/O 76535DC8 E739 776089D8 76539168
17 VDEVs  13 LINKs found
```

LOCATE Subcommand



Notes:

1 With no parameters, LOCATE will continue from the last LOCATE subcommand.

Purpose

The LOCATE subcommand scans the dump for a specified hexadecimal or character string.

Operands

hexchars
: is a string of hexadecimal digits, a maximum of 60 characters (30 bytes of data), in which the length must be a multiple of two.

'chars
: is a character string, a maximum of 60 characters (59 characters of data), that starts with a single quotation mark and cannot contain blanks. Use the *hexchars* option above if blanks are necessary.

L, R, or U
: is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *start*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

start
: is the one- to sixteen-significant digit hexadecimal start address for the search. This address can be sixteen significant characters.

end
: is the one- to sixteen-significant digit hexadecimal ending address of the search. The ending address must be larger than the start address. This address can be sixteen significant characters.

END
: uses the highest address in the dump as the ending address.

2Gig
: is the same as saying 80000000, just easier to type. If the area being searched is smaller than this size, the smaller size will be used.

inc
: is the one- to four-digit hexadecimal increment between compares. The default is 1. The maximum increment can be 1000.

SKIPS
generates a message for each page in the address range specified that is not in the dump.

Usage Notes

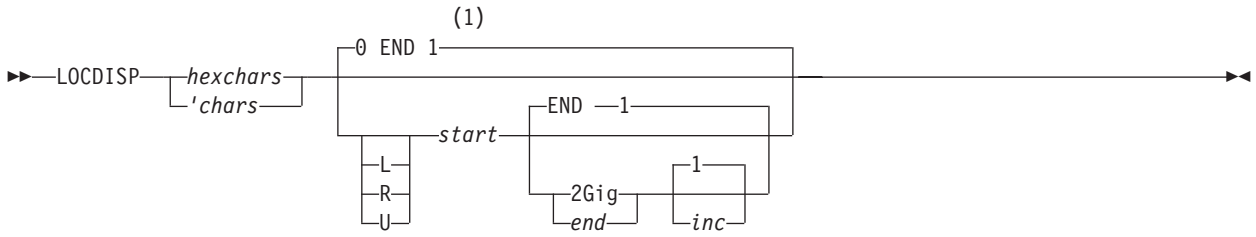
1. If entered with no parameters, LOCATE will restart where it stopped and find the next occurrence of the desired value, assuming the end address has not been reached. LOCATE with no parameters will continue in the same addressing mode that the first LOCATE used even if VMDTSET ADDRESS has been used in between to change the default.
2. LOCATE will display only the address at which the first occurrence of the desired value is found.
3. To locate all occurrences of the string, you must either repeat the LOCATE command (as described above) or use the LOCDISP macro.
4. When you use the *'chars* form of the input string, be sure to enter the string exactly as you want to search for it. For example, if you enter the string in lower case, and it appears in storage in upper case, it will not be found in your search and will not be displayed.
5. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
6. To halt a long-running LOCATE command, press the PA1 key to break out of full screen mode and then enter the HI command. After an HI, a summary line will be displayed to show the last address searched. A new LOCATE with no parameters may be entered to resume searching from this location.

Examples

Typical use of and output from the LOCATE subcommand:

```
>>> locate 0def LA803A0 00A813A0 2
STRING 0def found at address L00000000_00A808BA

>>> locate
STRING 0def found at address L00000000_00A808F0

>>> locate
STRING 0def found at address L00000000_00A80D4E

>>> locate
STRING 0def not found between L00000000_00A80D4E and 00000000_00A813A0

>>> locate
HCQLOC029E No successful LOCATE to resume

>>> locate e31010 R1813E8 001819E8 2
STRING e31010 found at address R00000000_00181424

>>> locate
STRING e31010 found at address R00000000_00181524

>>> locate
STRING e31010 found at address R00000000_00181660

>>> locate
STRING e31010 found at address R00000000_0018175C
```

LOCDISP Macro



Notes:

1 The default is all of real storage with an increment of 1. CAUTION: This could take a long time and generate a large amount of output.

Purpose

The LOCDISP macro scans the dump for a specified hexadecimal or character string and displays all occurrences of the string.

Operands

hexchars
: is a string of hexadecimal digits, a maximum of 60 characters (30 bytes of data), in which the length must be a multiple of two.

'chars
: is a character string, a maximum of 60 characters (59 characters of data), that starts with a single quotation mark and cannot contain blanks. Use the *hexchars* option above if blanks are necessary.

L, R, or U
: is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *start*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

start
: is the one- to sixteen-significant digit hexadecimal start address for the search. This address can be sixteen significant characters.

end
: is the one- to sixteen-significant digit hexadecimal ending address of the search. The ending address must be larger than the start address. This address can be sixteen significant characters.

END
: uses the highest address in the dump as the ending address.

2Gig
: is the same as saying 80000000, just easier to type. If the area being searched is smaller than this size, the smaller size will be used.

inc
is the one- to four-digit hexadecimal increment between compares. The default is 1.

Usage Notes

1. LOCDISP issues multiple LOCATE subcommands and finds all occurrences of the desired string between the start and end addresses. The storage around each occurrence is displayed.
2. You may enter HI to halt further execution. The data collected before the halt will be placed into the dump session.
3. When you use the *'chars* form of the input string, be sure to enter the string exactly as you want to search for it. For example, if you enter the string in lower case, and it appears in storage in upper case, it will not be found in your search and will not be displayed.

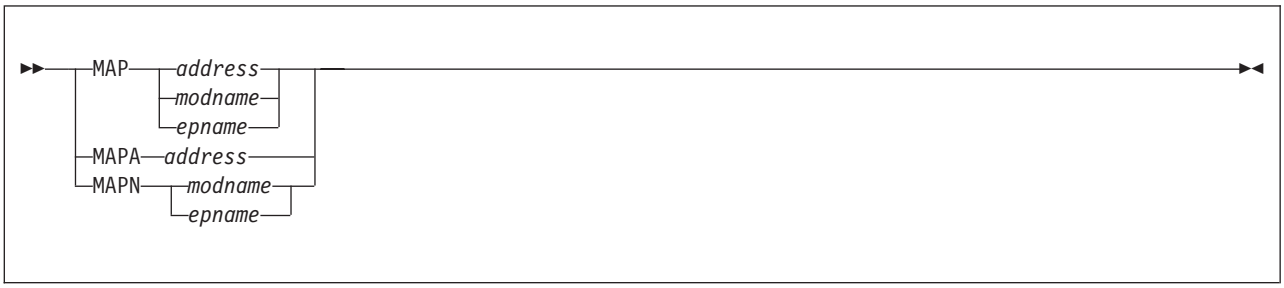
Examples

Typical use of and output from the LOCDISP macro:

```
>>> locdisp 0def A803A0 00A813A0 2

---> String 0def Found At L00000000_00A808BA = HCPLOG+51A
 00A808AA +0000 4710C516 91105001 47E0CA46 58F0C11C *..E.J.&..\...0A.*
 00A808BA +0010 0DEF4110 000158F0 C1200CEF 58E0C0D4 *.......0A....\{M*
 00A808CA +0020 58F0C124 *.0A.*

---> String 0def Found At L00000000_00A808F0 = HCPLOG+550
 00A808E0 +0000 100058E0 C0F858F0 C12C0DEE 58F0C130 *...\{8.0A....0A.*
 00A808F0 +0010 0DEF58E0 C0D458F0 C1240DEE 47F0C570 *...\{M.0A....0E.*
 00A80900 +0020 91405003 *J &.*

---> String 0def Found At L00000000_00A80D4E = HCPLOG+9AE
 00A80D3E +0000 FFF158E0 C0D458F0 C1940DEE 58FD0014 *.1.\{M.0AM......*
 00A80D4E +0010 0DEF58E0 C0D458F0 C1980DEE 1F005000 *...\{M.0AQ....&.*
 00A80D5E +0020 D02007F9 *}..9*

String 0def not found between L00000000_00A80D4E and 00000000_00A813A0
3 occurrences found
```

MAP Subcommand

PI



Purpose

The MAP subcommand is used to convert addresses to entry point names or module offsets, and vice versa.

Operands

address
: is the one- to eight-significant digit hexadecimal logical address in the dump. If it lies within a module, the module name and offset within that module will be displayed.

modname
: is the one- to eight-character name of a module. If the module is in the dump, the logical address of the module will be displayed.

epname
: is the one- to eight-character name of an entry point within a module. If the entry point is in the dump, the logical address of the entry point will be displayed.

Usage Notes

1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. When the input to MAP is a module or entry point name, the output is an address of the form *nnnnnnnn_nnnnnnnn*, regardless of the setting of SET HIWORD. This is to make it easier for macros to predict the output format.
3. EXTRACT MAPN and EXTRACT MAPA can be used from a macro as an alternative. For further information, see "EXTRACT Subcommand" on page 60.

Examples

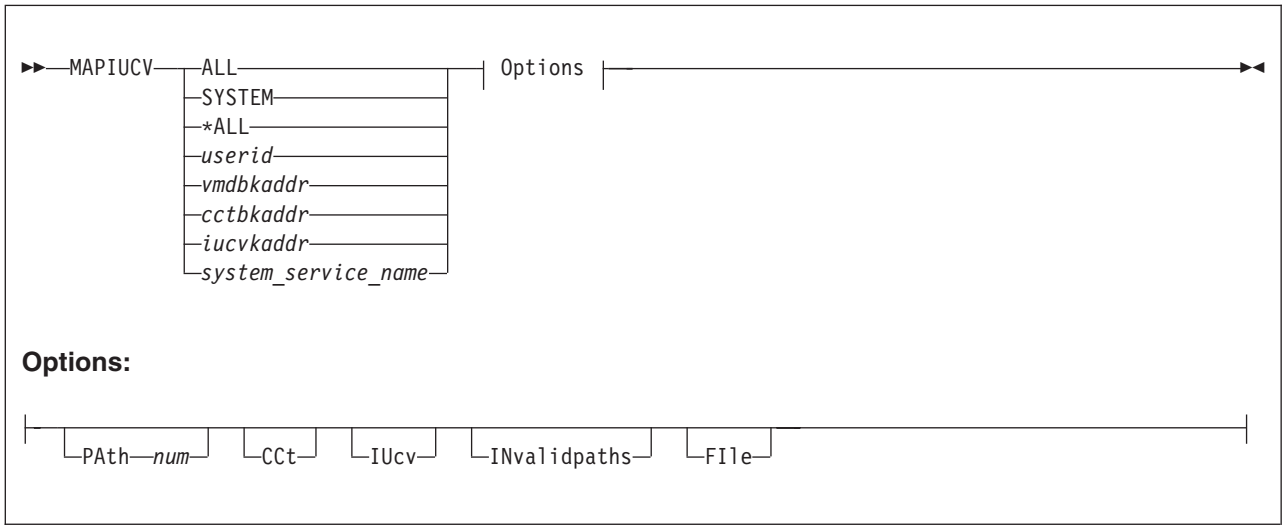
Typical use of and output from the MAP subcommand:

```
>>> map hcpsvcaa
HCPSVCAA is at address 00000000_002E6270

>>> map 002E6270
00000000_002E6270 is at HCPSVCAA

>>> map hcplog
HCPLOG is at address 00000000_00A803A0
```

```
>>> map A803A0
00000000_00A803A0 is at HCPLOG

>>> mapa A803A0
00000000_00A803A0 is at HCPLOG

>>> mapn hcplog
HCPLOG is at address 00000000_00A803A0
```

PI **end**

MAPIUCV Macro



Purpose

The MAPIUCV macro is used to display IUCV-related control blocks for one or more users or system services.

Operands

ALL
specifies all users.

SYSTEM
***ALL**
specifies all system services.

userid
specifies the user ID of a user to display.

vmdbkaddr
specifies a one- to eight-significant digit hexadecimal logical address of the single user or system service associated with the VMDBK.

cctbkaddr
specifies a one- to eight-significant digit hexadecimal logical address of the single user or system service associated with the CCTBK.

iucvbkaddr
specifies a one- to eight-significant digit hexadecimal logical address of the single user or system service associated with the IUCVBK

system_service_name
specifies the name of a system service to display. Allowable values are:
*ACCOUNT
*ASYNCMD
*BLOCKIO
*CCS
*CONFIG
*CRM
*IDENT

*LOGREC
*MONITOR
*MSG
*MSGALL
*RPI
*SCLP
*SIGNAL
*SPL
*SYMPTOM
*VMEVENT
*VSWITCH

PAth *num*
limits the output to only the path specified. *num* is a one- to four-digit hexadecimal value of the path to be used.

CCt
specifies that the relevant CCTs should be displayed.

IUcv
specifies that the relevant IUCVBKs should be displayed

INvalidpaths
includes information about invalid paths; these paths are usually suppressed.

FIle
causes the output to be placed in a file called "dumpname MAPIUCV A1". If the file already exists, the new information will be appended to in the existing file.

Usage Notes

1. The MAPIUCV macro can do a lot of analysis of IUCV-related information, but it can also generate a large amount of output.
2. You may use HI (Halt Interpretation) to terminate processing and get partial results.
3. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro will be replaced by the value pointed to by the cursor.
4. This macro is supported only for CP dumps.

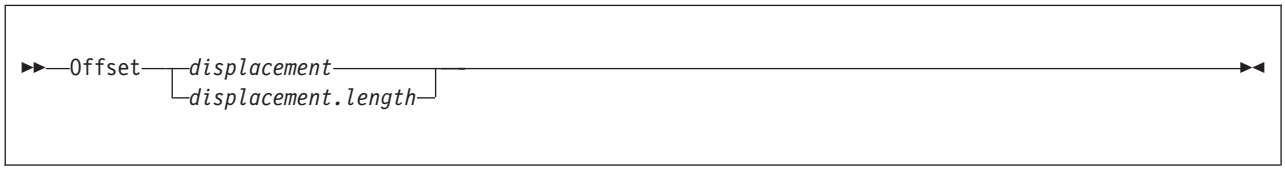
Examples

Typical use of and output from the MAPIUCV macro:

```
>>> mapiucv *all
Summary of System Services
  *MSGALL      ** no iucv **
  *MSG      IUCV 01DAF340  CCT 01DBC7B0  Msgct 0
  *BLOCKIO     ** no iucv **
  *RPI         ** no iucv **
  *MONITOR     ** no iucv **
  *SIGNAL   IUCV 00C87050  CCT 00D3F670  Msgct 0
  *CCS      IUCV 01E210C0  CCT 01E517B0  Msgct 0
  *SPL         ** no iucv **
  *SYMPTOM  IUCV 00CF5050  CCT 00D2C5D0  Msgct 0
  *ACCOUNT  IUCV 00CF7050  CCT 00D3C0A8  Msgct 0
  *LOGREC   IUCV 00CF6050  CCT 00D36BC0  Msgct 0
  *CRM         ** no iucv **
  *IDENT    IUCV 00C48050  CCT 00CF77B0  Msgct 0
  *CONFIG      ** no iucv **
  *VSWITCH  IUCV 01E38388  CCT 01E382B0  Msgct 0
```

```
>>> mapiucv *msg
The CSS entry for *MSG is at HCPIUG+00000040  = 00A78128:
   _00A78128 +0000 5CD4E2C7 40404040 028000FF 00248DC0 **MSG    .......{*
   _00A78138 +0010 00000000 00248EC0 00249030 00249060 *.......{.......-*
   _00A78148 +0020 01DAF340 FFFF0000 00000000 00000000 *..3 ............*
   _00A78158 +0030 00000000 00000000 00000000 00000000 *................*
IUCV 01DAF340  CCTBK 01DBC7B0  VMDBK 00002000  user/svc *MSG
Max path# is 0007, max path# allowed is FFFF
Path PDENT@   TPth Status   TgtCCT@  TgtVMD@  TgtUser  Status
   0 01DBC698 0001 C0000000 01DB0C38 01DA7000 RSCS     Valid
   1 01DBC6B8 0000 C0000000 01DEF888 01DED000 PVM      Valid
8 PDEs found, 6 are invalid

>>> mapiucv operator
Input parameter is a userid
IUCV 01D38F40  CCTBK 01D38EE0  VMDBK 017BB000  user/svc OPERATOR
Max path# is 0007, max path# allowed is 0040
Path PDENT@   TPth Status   TgtCCT@  TgtVMD@  TgtUser  Status
8 PDEs found, all are invalid
```

OFFSET Subcommand



Purpose

The OFFSET subcommand is similar to the DISPLAY subcommand. The difference is that it gets the address to display from an offset into the data from the previous display.

Operands

displacement
: is the one- to four-digit hexadecimal offset into the data from the previous display subcommand. The fullword at that offset is used as a pointer to the data to be displayed by this subcommand.

length
: is the one- to four-digit hexadecimal length of data to display. This may be specified as a hexadecimal number, a register, in the form (Rx), or a register plus hexadecimal offset in the form (Rx+yyy). If the length is 0, then approximately one full screen of data will be displayed.

Usage Notes

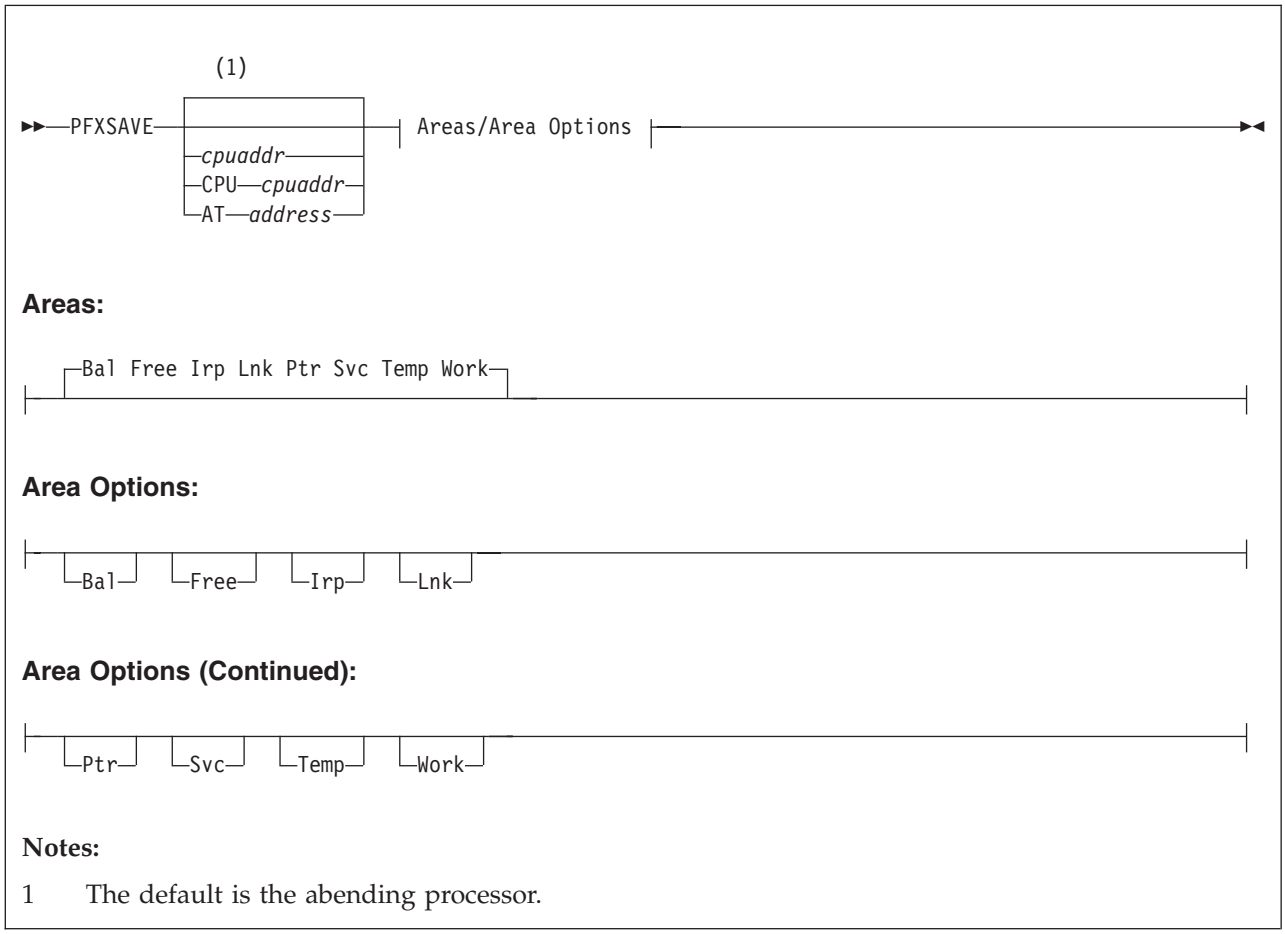
The addressing mode (logical, real or user) used on the previous DISPLAY will be used by the OFFSET subcommand even if the default has been changed since the DISPLAY was issued.

Examples

Typical use of and output from the OFFSET subcommand:

```
>>> display 770B47C0.40
_770B47C0 +0000 00000000 00000000 00000000 00000000 *................*
_770B47D0 +0010 00000000 0000007A 76D3DF38 00000000 *.......:.L......*
_770B47E0 +0020 0000006A 00000208 0468E780 00000000 *...]......X.....*
_770B47F0 +0030 00000000 00000000 00000000 00000000 *................*

>>> offset 28.10
_0468E780 +0000 00000000 00000000 00000000 00000000 *................*
```

PFXSAVE Macro



Purpose

The PFXSAVE macro displays Prefix Static Saveareas for a selected processor.

Operands

cpuaddr

CPU *cpuaddr*
is the one- to four-digit hexadecimal CPU address of the processor for which the saveareas to be displayed. The default is the abending processor.

AT *address*
is the one- to eight-significant digit hexadecimal logical address of the storage to be interpreted as a Prefix Page.

Bal
Requests selection of the BALSV area.

Free
Requests selection of the FRESV area.

Irp
Requests selection of the IRPSV area.

Lnk
Requests selection of the LNKSV area.

Ptr
Requests selection of the PTRSV area.

Svc
Requests selection of the SVCSV area.

Temp
Requests selection of the TMPSV area.

Work
Requests selection of the WRKSV area.

Usage Notes

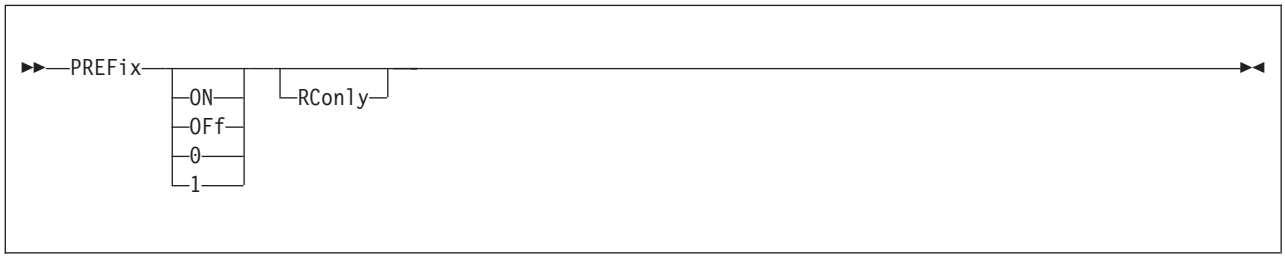
1. If one or more saveareas are selected by name, only those selected will be displayed. If none are selected, then all saveareas will be displayed.
2. The input for this macro can be in any order.
3. The CPEBK subcommand is used to display the individual save areas. It respects the settings of SET REG MAP, SET MAP HIWORD.
4. This macro is supported only for CP dumps.

Examples

Typical use of and output from the PFXSAVE macro:

```
>>> pfxsave fre
---> FREE SAVE at 00000700 CPUNO 0000 (failing)
 SVGFPNT 00000000  SVGBPNT 00000000  SVGSFQP 00000000  SVGCPRQ 00000000
 SVGSCHC 00  SVGCALC 00  SVGFORM E0  SVGRETN 00000000 (64-bit)
R0 00000000_00000000                 R8 00000000_090A6800
R1 00000000_000000C0                 R9 00000000_090A6800
R2 00000000_00000000                 RA 00000000_01DED000
R3 00000000_00000051                 RB 00000000_01DED000
R4 00000000_00000000                 RC 00000000_00274380 HCPPAG
R5 00000000_0000E000 HCPSYSLC        RD 00000000_00C65A00
R6 00000000_002B0830 HCPRSMCM        RE 00000000_80274642 HCPPAG+2C2
R7 00000000_00185210 HCPHTU+5C0      RF 00000000_00000000
SVGWRK0-1 00000000 00000000 00000000 017883D8 *.............cQ*
SVGWRK2-3 00000000 80000000 00000000 000001FD *................*
SVGWRK4-5 00000000 002B1E40 00000000 00000000 *....... ........*
SVGWRK6-7 00000000 80A6484C 00000000 00000000 *.....w.<........*
SVGWRK8-9 00240637 4C4C4C4C 00000000 00000016 *....<<<<........*

>>> pfxsave cpu 2 svc
---> SVC SAVE at 00B6C840 CPUNO 0002
_00B6C840 +0000 00000000 00000000 00000000 00D63CE0 *.............O.\*
_00B6C850 +0010 00000000 00000000 00000000 00000079 *.............`*
_00B6C860 +0020 00000000 00000000 00000000 0000E000 *.............\.*
_00B6C870 +0030 00000000 00D639F0 00000000 002B2A28 *.....O.0........*
```

PREFIX Subcommand



Purpose

The PREFIX subcommand controls the use of prefixing low storage of the failing processor within the VM Dump Tool. The PREFIX subcommand is the opposite of the ABSOLUTE command.

Operands

(null)
causes the current setting to be displayed.

ON
PI 1
specifies that a reference to the prefix page(s) is treated as a real address as seen on the abending processor. That is, the prefix register of the abending processor is used in resolving such addresses. **PI end**

OFf
PI 0
sets address prefixing off. References to the prefix page(s) are treated as absolute addresses, and no prefixing is applied. **PI end**

PI RConly
specifies that an indication of the setting at entry should be returned as a return code. **PI end**

Usage Notes

1. In a 390-mode system, the prefix page is page 0, storage addresses 0-4095 (X'0' - X'FFF'). In 64-bit mode, the prefix page consists of pages 0 and 1, addresses 0-8191 (X'000' - X'1FFF').
2. Note that the ABSOLUTE command is the opposite of the PREFIX command. Setting ABSOLUTE ON is the same as setting PREFIX OFF. They affect the same setting and may be used interchangeably. See also the “ABSOLUTE Subcommand” on page 27 for related information.
3. Using the '0', '1', or 'RConly' parameters causes all output to the dump session to be suppressed. This is useful inside a macro for easily saving, setting, and then restoring the PREFIX status.
4. The VMDTSET and VMDTQRY subcommands are an alternate way to set and determine the prefix setting.

Examples

The following sequence shows the interaction of and the output from the PREFIX and ABSOLUTE commands:

```
>>> absolute
Absolute addressing off

>>> prefix
Prefixing of Page 0 is on

>>> absolute on
Absolute addressing on

>>> prefix
Prefixing of Page 0 is off

>>> prefix on
Prefixing of Page 0 is on

>>> absolute
Absolute addressing off

>>> prefix
Prefixing of Page 0 is on

>>> prefix off
Prefixing of Page 0 is off

 PI
>>> prefix on rconly
```

(return code from above is 0)

```
>>> prefix 0 PI end
```

PREG Subcommand

Notes:

1 The default is the abending processor.

Purpose

The PREG subcommand displays the prefix register from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

Usage Notes

You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.

Examples

Typical use of and output from the PREG subcommand:

```
>>> preg
  PFX VALUE 04508000

>>> preg 3
  Data for CPU 0003

  PFX VALUE 765EC000
```

PSWS Subcommand

Notes:

1 The default is the abending processor.

Purpose

The PSWS subcommand displays the new and old PSWs from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

Usage Notes

1. You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
2. The current PSW is also displayed if the dump is a VMDUMP or Stand Alone Dump.

Examples

Typical use of and output from the PSWS subcommand:

```
>>> psws
  Restart  Old FFFFFFFF FFFFFFFF
           New 00FC3000 8467B4F8 HCPSVFDU
  External Old 070E4000 80000000 HCPPFX
           New 000C0000 845A6978 HCPEXTEX
  SVC      Old 000C1000 845AAE54 HCPFRE+B54
           New 000C0000 8467B3A8 HCPSVFD0
  Program  Old 000C0000 84747580 HCPQCO+580
           New 000C0000 846447A8 HCPPRGIN
  Machine  Old 00000000 00000000
           New 00080000 8460A800 HCPMCHIN
  I/O      Old 070E4000 80000000 HCPPFX
           New 000C0000 845CDBB8 HCPIFIIN

>>> psws
Restart  Old 00000000 00000000 00000000 00000000
         New 00F43000 80000000 00000000 002DD198 HCPSVFDU
External Old 03041000 80000000 00000000 002AB044 HCPRUN+44
         New 00040000 80000000 00000000 00124028 HCPEXTEX
SVC      Old 00040000 80000000 00000000 001A0A48 HCPIOL+A48
         New 00040000 80000000 00000000 002DD028 HCPSVFD0
Program  Old 00042000 80000000 00000000 0013617A HCPFRE+17A
```

```
         New 00040000 80000000 00000000 0027C028 HCPPRGIN
Machine  Old 00000000 00000000 00000000 00000000
         New 00000000 80000000 00000000 001F4238 HCPMCHIN
I/O      Old 03041000 80000000 00000000 000FD0D6 HCPDSP+D6
         New 00040000 80000000 00000000 00188028 HCPIFIIN
```

QCPLEVEL Macro

Purpose

The QCPLEVEL macro is used to simulate a QUERY CPLEVEL command from the information in the dump.

Usage Notes

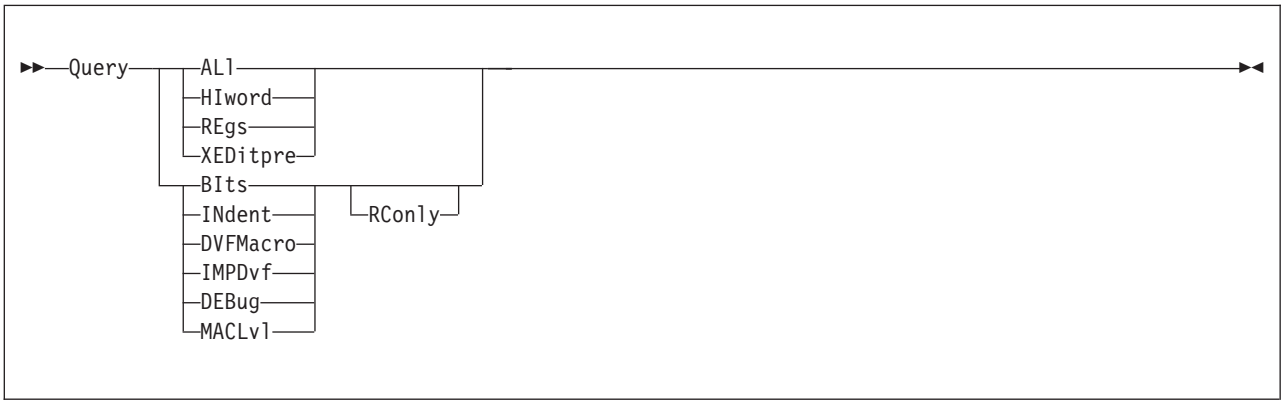
This macro is supported only for CP dumps

Examples

Typical output from the QCPLEVEL macro:

```
qcplevel
z/VM Version 5 Release 4.0, service level 0000 (CP 64-BIT)
Generated at 07/01/08 17:02:16.000000, IPLd at 07/01/08 17:14:55.144533
```

QUERY Subcommand



Purpose

The QUERY subcommand is used to determine the settings of a number of user-settable variables.

The VMDTQRY subcommand is recommended over the QUERY subcommand. All operands of QUERY that have been supported in the past will continue to be supported without change. New operands will be added to VMDTQRY only.

Operands

ALl
specifies that all SET values are to be displayed.

HIword
Returns the current setting of the HIWord variable. This variable is used to determine how to display double-word addresses when the high order full-word of the address is zero.

Responses include:

FULL
the display will consist of the full double-word address with an underscore (_) between the two words.

BLANK
the display will consist of only the underscore (_) and the second full-word.

SINGLE0
the display will consist of a single zero, the underscore (_), and the second full-word.

REgs
Indicates how register values are to be displayed from the GREG subcommand.

Responses include:

MAP
the value of each register is mapped to a module and displacement.

NOMAP
Register values are not to be mapped to modules.

SHORT
Register values should be assumed to be 4 bytes long.

LONG
Register values should be assumed to be 8 bytes long.

XEDitpre
displays the current setting for the XEDIT prefix string.

BIts
Indicates whether bit definitions are to be displayed automatically with VM Dump Tool subcommands including RDEV, VDEV, and a number of trace types.

Responses include:

ON bit and code meanings will be displayed.

OFF
bit and code meanings will not be displayed

INDENT
displays or returns the current indent setting.

DVFMacro
specifies that the value of the SET DVFMACRO input is to be returned.

IMPDvf
specifies that the value of the SET IMPDV input is to be returned.

DEBug
displays or returns the current DEBUG setting.

MACLvl
specifies that the macro nesting level should be displayed. The response from issuing the QUERY command is 0. When issued from a macro, it is 1 or greater to indicate the nesting level.

PI RConly
returns the result as a return code. RC=0 indicates OFF; RC=1 indicates ON. **PI end**

Usage Notes

1. See also the "SET Subcommand" on page 130 for more information.
2. See also the "INDENT Subcommand" on page 87 for more information.
3. For QUERY XEDITPRE, the output is normally displayed as uppercase chacacters. If you enter input using the Xnn notation, the X is displayed as lowercase to indicate the input is in the Xnn format and is not a string starting with X.

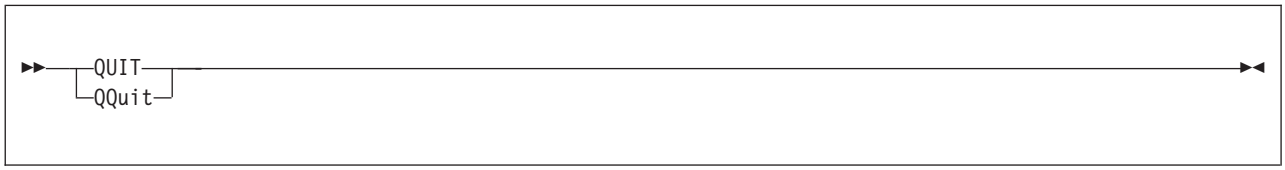
Examples

Typical output from the QUERY subcommand:

```
>>> query regs
REGS    set to  NOMAP SHORT

>>> query hiword
HIWORD  set to  FULL

>>> query bits
BITS    set to  ON

>>> query indent
```

```
INDENT   set to  0

>>> set xed /
complete
>>> query xed
XEDITPRE set to /
>>> query all
REGS     set to NOMAP SHORT
HIWORD   set to FULL
BITS     set to ON
INDENT   set to  0
DVFMACRO set to OFF
IMPDVF   set to OFF
MACLVL   set to  0
XEDITPRE set to  /
DEBUG    set to OFF

>>> set xed yyy
complete
>>> query all
REGS     set to NOMAP SHORT
HIWORD   set to FULL
BITS     set to ON
INDENT   set to  0
DVFMACRO set to OFF
IMPDVF   set to OFF
MACLVL   set to  0
XEDITPRE set to YYY
DEBUG    set to OFF

>>> set xed xff
complete
>>> query xed
XEDITPRE set to xFF

>>> set xeditpre off
complete
>>> query xed
XEDITPRE set to OFF
```

QUIT/QQUIT Subcommand



Purpose

The QUIT subcommand terminates the VM Dump Tool and does not save the dump log file.

Operands

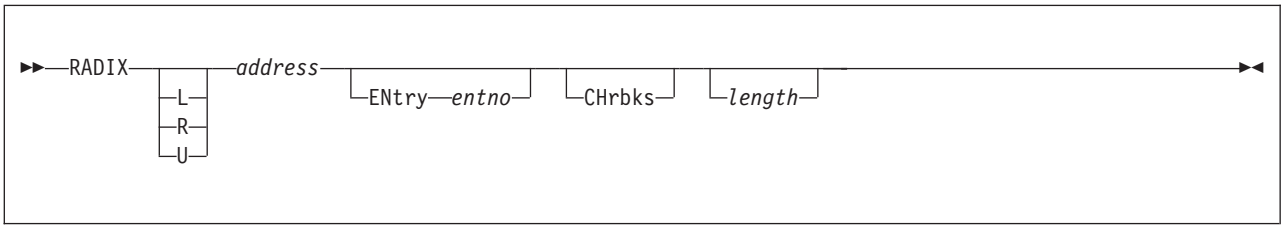
QUIT
terminates unless there are changes, which will give you a warning.

QQUIT
discards any changes and terminates.

Usage Notes

See also the "FILE/FFILE Subcommand" on page 70 and the "SAVE/SSAVE Subcommand" on page 127 for related functions.

RADIX Macro



Purpose

The RADIX macro displays the contents of, or data areas pointed to, by a radix tree. Radix trees are used to map devices, CP Exits, and other data areas in CP.

Operands

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *address*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

address
specifies the one- to sixteen-significant digit hexadecimal address of the anchor of this radix tree.

ENtry *entno*
specifies that only a single entry is to be processed instead of all of the active entries in the radix tree. *entno* is the one- to four-digit hex number of the entry to display.

CHrbks
specifies that the CHRBKs themselves that make up the radix tree should be displayed.

length
specifies the one- to four-digit hexadecimal length of data that should be displayed for each entry found in the lowest level of the radix tree.

Usage Notes

1. The address specified is not checked for being a CHRBK.
2. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.

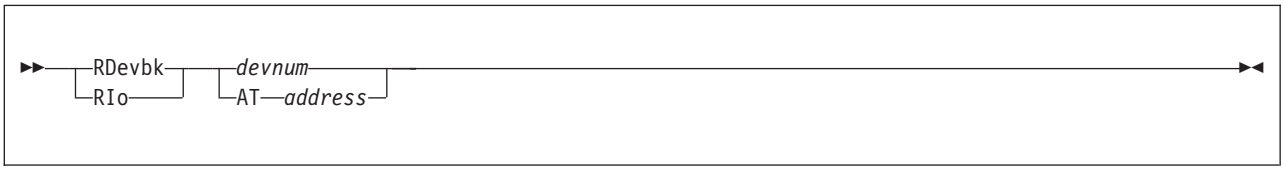
Examples

Typical use of and output from the RADIX macro:

```
>>> radix 770B4700 entry 1001
Lvl Idx @CHRBK   displ entry
 L1   1 770B4700 00004 765385F0
 L2   0 765385F0 00000 76538598
 L3   0 76538598 00000 76538540
 L4   1 76538540 00004 7653D410
Element 1001 contains  7653D410
```

```
>>> radix 770B4700 entry 1001 chrbk
L0 ---> CHRBK at 770B4700 (element 1)
  00000000_770B4700 76538490 765385F0 770B4EC0 77631608 *..d...e0..+{....*
  00000000_770B4710 00000000 00000000 60000010 0BBB0011 *........-.......*
  00000000_770B4720 00004000 00000000 00000000 00000000 *................*
  00000000_770B4730 00000000 00000001 00000000 00000000 *................*
  00000000_770B4740 00000000 00000000 00000000 0000010F *................*
L1 ---> CHRBK at 765385F0 (element 0)
  00000000_765385F0 76538598 00000000 00000000 00000000 *..eq............*
  00000000_76538600 00000000 00000000 00000000 00000000 *................*
  00000000_76538610 00000000 00000000 00000000 00000000 *................*
  00000000_76538620 00000000 00000000 00000000 00000000 *................*
  00000000_76538630 4CC3C8D9 4C4C4C4C 800007CA 00E2C3D5 *<CHR<<<<.....SCN*
L2 ---> CHRBK at 76538598 (element 0)
  00000000_76538598 76538540 76538330 00000000 00000000 *..e...c.........*
  00000000_765385A8 7653D568 7653D310 00000000 00000000 *..N...L.........*
  00000000_765385B8 00000000 00000000 76539110 76538028 *.........j.....*
  00000000_765385C8 00000000 00000000 00000000 00000000 *................*
  00000000_765385D8 4CC3C8D9 4C4C4C4C 800007CA 00E2C3D5 *<CHR<<<<.....SCN*
L3 ---> CHRBK at 76538540 (element 1)
  00000000_76538540 765384E8 7653D410 00000000 00000000 *..dY..M.........*
  00000000_76538550 00000000 00000000 00000000 00000000 *................*
  00000000_76538560 00000000 00000000 00000000 00000000 *................*
  00000000_76538570 00000000 00000000 00000000 00000000 *................*
  00000000_76538580 4CC3C8D9 4C4C4C4C 800007CA 00E2C3D5 *<CHR<<<<.....SCN*
Element 1001 contains  7653D410
```

RDEVBK Subcommand



Purpose

The RDEVBK subcommand locates, analyzes, and displays a real device block.

Operands

devnum
is the one- to four-digit hexadecimal real device number of the device. The RDEVBK is located by a scan of the real device radix tree.

AT *address*
is the one- to eight- significant digit hexadecimal logical address in the dump of the RDEVBK to be displayed.

Usage Notes

1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor
2. This subcommand is supported only for CP dumps.

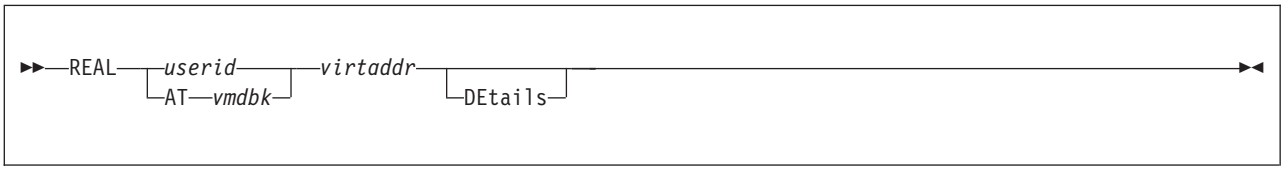
Examples

Typical use of and output from the RDEVBK:

```
>>> rdev 0600
Device number 0600 Subchannel number 000B
RDEVBK          at 00E03BA8
Active IORBK    at 00D8FC08
Channel Program at 0FA9B538

>>> rdev at 00E03BA8
Device number 0600 Subchannel number 000B
RDEVBK          at 00E03BA8
Active IORBK    at 00D8FC08
Channel Program at 0FA9B538

>>> dts bits on
complete

>>> rdev 0600
  Bits  defined in RDEVCLAS         (04)
    04 CLASDASD DIRECT ACCESS STORAGE DEVICE CLASS
  Bits  defined in RDEVDFLG         (82)
    80 RDEVAUTO 370X - AUTO LOAD/DUMP ACTIVE
    02 RDEVMDCP DASD - Caching in MDC enabled for devices with MDC
       (DFLTOFF) See RDEVHSID for explanation.
    80 RDEVPSUP TERM - PRINT SUPPRESS AVAILABLE
  Bits  defined in RDEVFEAT         (80)
    80 FTROPRDR GRAF - OPERATOR ID CARD READER
    80 FTR4WCGM SPOL - 3800 WITH FOUR WRITEABLE CHARACTER GENERATI
       MODULES
```

```
    80 FTR7TRK TAPE - 7-TRACK FEATURE
    80 FTRRPS DASD - ROTATIONAL POSITIONAL SENSING
    80 FTRTERM SPEC - UNSUPPORTED TERMINAL DEVICE
Device number 0600 Subchannel number 000B
RDEVBK          at 00E03BA8
Active IORBK    at 00D8FC08
Channel Program at 0FA9B538
```

REAL Macro



Purpose

The REAL macro displays the real address corresponding to a guest's virtual address. The result is the equivalent of the LRA instruction.

Operands

userid
is the user ID of the guest, or SYSTEM.

AT *vmdbk*
is the one- to eight-significant digit hexadecimal logical address in the dump of the VMDBK for which the real address is to be found and displayed.

virtaddr
is the one- to sixteen-significant digit hexadecimal host virtual address (guest real address) to be translated.

DEtails
indicates that intermediate information about the translation should be provided.

Usage Notes

1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. This macro is supported only for CP dumps.

Examples

Typical use of and output from the REAL macro:

```
>>> real operator 10f3
data address is 00000000_094240F3

>>> real at 017BB000 10f3
data address is 00000000_094240F3

>>> real operator 10f3 details
breakout of input address - 00000000000010F3 - (Bits,Hex,Table Entry Offset)
        Region      Region      Region
        First       Second      Third       Segment    Page       Byte
        Index       Index       Index       Index      Index      Index
      ----------- ----------- ----------- ----------- -------- ------------
Bits: 00000000000 00000000000 00000000000 00000000000 00000001 000011110011
Hex:          000         000         000         000       01          0F3
TEOf:        0000        0000        0000        0000     0008

address           ASCE             Table Origin     ..GPSXR. DT TL
00000000_017BB768 000000000EC6A000 000000000EC6A000 ..00000. 00 00 (STD)
```

```
address           STE contents     Table Origin     O.P...IC TT ..
00000000_0EC6A000 000000000ECDA800 000000000ECDA800 1.0...00 00 .. (STE)

address           PTE contents     PFRA             .IP..... .. .L
00000000_0ECDA808 0000000009424000 0000000009424000 .00..... .. .0 (PTE)

data address is 00000000_094240F3
```

REGISTER Subcommand

Notes:

1 The default is the abending processor.

Purpose

The REGISTER subcommand displays the register and control information from the specified processor.

Operands

CPU *cpuaddr*
cpuaddr
is a one- to four-digit hexadecimal CPU address. The default is the abending processor.

Usage Notes

1. The registers are displayed in the following order as if the individual subcommands had been entered. The individual subcommands are listed in parentheses below. Display order is:
   a. General registers (Gregs)
   b. Control registers (CRegs)
   c. Access registers (Aregs)
   d. Floating point registers (FRegs)
   e. Clocks and Comparators (CLocks)
   f. Prefix register (Preg)
   g. PSWs (PSws).

   For further information about the output, see the help for each subcommand.
2. You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.

Examples

```
>>> regs
R0 00000000_00000000                R8 00000000_8002FC60
R1 00000000_01D16820                R9 00000000_8002FC60
R2 00000200_0003DAA0 HCPOPR+1BB0    RA 00000000_002B0830 HCPRSMCM
R3 00000000_002B0A98 HCPRSMAQ       RB 00000000_01DED000
R4 00000000_8002C080 HCPWRK+19080   RC 00000000_00062520 HCPALF
R5 00000000_01D16000                RD 00000000_0177B600
R6 00000000_000633E0 HCPALF+EC0     RE 00000000_09CB8F28
R7 00000000_00185210 HCPHTU+5C0     RF 00000000_80200010 HCPMCS+4B0

C0-3 00000000_1400FE40 00000000_0000A007 00000000_0F9FE080 00000000_00000000
```

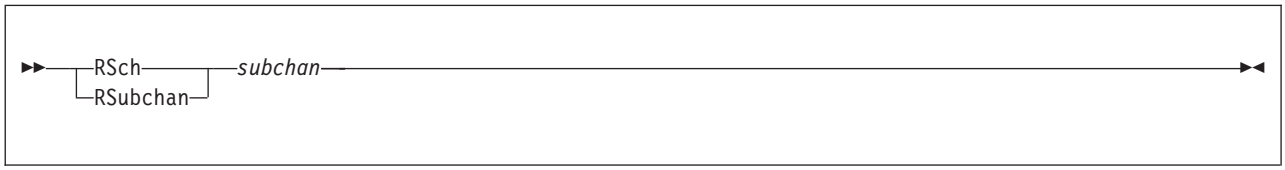
```
C4-7 00000000_00000000 00000000_0F9FE000 00000000_80000000 00000000_0C812100
C8-B 00000000_00000000 00000000_00000000 00000000_00000000 00000000_00000000
CC-F 00000000_0F280161 00000000_0000A007 00000000_1F000000 00000000_00000000

A0-7  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A8-15 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Floating Point Control Register 00000000
F0-3   0000000000000000  0000000000000000  0000000000000000  0000000000000000
F4-7   0000000000000000  0000000000000000  0000000000000000  0000000000000000
F8-B   0000000000000000  0000000000000000  0000000000000000  0000000000000000
FC-F   0000000000000000  0000000000000000  0000000000000000  0000000000000000

TOD Clock BCBBA564 CE6CE1AE   Clock Comparator BCBBA564 FC8DE560

CPU Timer 00000000 00D228A0

PFX VALUE 00030000

Restart  Old 04046001 80000000 00000000 00062F2A HCPALF+A0A
         New 04F4F000 80000000 00000000 002FCE20 HCPSVF@1
External Old 07041000 80000000 00000000 00124F82 HCPDSP+282
         New 0404C000 80000000 00000000 001456A8 HCPEXTEX
SVC      Old 04042000 80000000 00000000 001A2284 HCPIOE+364
         New 0404C000 80000000 00000000 002FCCF8 HCPSVF@0
Program  Old 04042000 80000000 00000000 00145726 HCPEXT+C6
         New 0404C000 80000000 00000000 002868D0 HCPPRGIN
Machine  Old 00000000 00000000 00000000 00000000
         New 00000000 80000000 00000000 001FA168 HCPMCHIN
I/O      Old 07064000 80000000 00000000 00000000
         New 0404C000 80000000 00000000 00195E30 HCPIFIIN
```

The following is typical for a 64-bit dump:

```
>>> regs
R0-3 00000000_000000F0 00000000_FFFF0000 00000000_00000000 00000000_00000000
R4-7 00000000_01EFAEC0 00000000_01EFAEC0 00000000_00000000 00000000_05000E78
R8-B 00000000_01F5EDA0 00000000_801A0624 00000000_00000000 00000000_01EFA000
RC-F 00000000_001A0000 00000000_01F01000 00000000_802A0268 00000000_001A0610

C0-3 00000000_1401EE40 00000000_0080D003 00000000_01F72040 00000000_00000000
C4-7 00000000_00000000 00000000_01F72000 00000000_C8000000 00000000_01D6D100
C8-B 00000000_00000001 00000000_00000000 00000000_00000000 00000000_00000000
CC-F 00000000_01FCFA01 00000000_01F5F100 00000000_1F000000 00000000_00000000

A0-7   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

F0-3   0000000000000000  0000000000000000  0000000000000000  0000000000000000
F4-7   0000000000000000  0000000000000000  0000000000000000  0000000000000000
F8-B   0000000000000000  0000000000000000  0000000000000000  0000000000000000
FC-F   0000000000000000  0000000000000000  0000000000000000  0000000000000000

TOD CLOCK B343B0DA B63FF504  TOD CLOCK COMP B343B0DA 0266FC00

CPU TIMER 00000000 00DDC400

PFX VALUE 00014000

Restart  Old 00000000 00000000 00000000 00000000
         New 00F43000 80000000 00000000 002DD198 HCPSVFDU
External Old 03041000 80000000 00000000 002AB044 HCPRUN+44
         New 00040000 80000000 00000000 00124028 HCPEXTEX
SVC      Old 00040000 80000000 00000000 001A0A48 HCPIOL+A48
         New 00040000 80000000 00000000 002DD028 HCPSVFD0
Program  Old 00042000 80000000 00000000 0013617A HCPFRE+17A
         New 00040000 80000000 00000000 0027C028 HCPPRGIN
```

```
Machine  Old 00000000 00000000 00000000 00000000
         New 00000000 80000000 00000000 001F4238 HCPMCHIN
I/O      Old 03041000 80000000 00000000 000FD0D6 HCPDSP+D6
         New 00040000 80000000 00000000 00188028 HCPIFIIN
```

RSCH Subcommand



Purpose

The RSCH subcommand locates, analyzes, and displays a real device block.

Operands

subchan
is the one- to four-digit hexadecimal real subchannel number. The RDEVBK is located by scanning the RDEVBKs sequentially until the corresponding subchannel is found.

Usage Notes

1. You may specify *subchan* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
2. This subcommand is supported only for CP dumps.

Examples

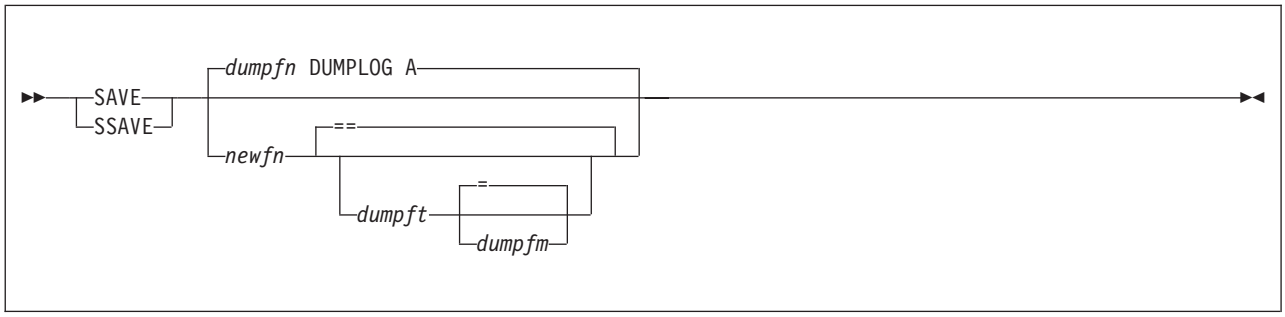
Typical output from the RSCH subcommand:

```
>>> rsch 000B
Device number 0600 Subchannel number 000B
RDEVBK          at 00E03BA8
Active IORBK    at 00D8FC08
Channel Program at 0FA9B538

>>> dts bits on
complete

>>> rsch 000B
  Bits  defined in RDEVCLAS         (04)
    04 CLASDASD DIRECT ACCESS STORAGE DEVICE CLASS
  Bits  defined in RDEVDFLG         (82)
    80 RDEVAUTO 370X - AUTO LOAD/DUMP ACTIVE
    02 RDEVMDCP DASD - Caching in MDC enabled for devices with MDC
       (DFLTOFF) See RDEVHSID for explanation.
    80 RDEVPSUP TERM - PRINT SUPPRESS AVAILABLE
  Bits  defined in RDEVFEAT         (80)
    80 FTROPRDR GRAF - OPERATOR ID CARD READER
    80 FTR4WCGM SPOL - 3800 WITH FOUR WRITEABLE CHARACTER GENERATI
       MODULES
    80 FTR7TRK TAPE - 7-TRACK FEATURE
    80 FTRRPS DASD - ROTATIONAL POSITIONAL SENSING
    80 FTRTERM SPEC - UNSUPPORTED TERMINAL DEVICE
Device number 0600 Subchannel number 000B
RDEVBK          at 00E03BA8
Active IORBK    at 00D8FC08
Channel Program at 0FA9B538
```

SAVBK Subcommand

SAVBK is a synonym for CPEBK. See “CPEBK Subcommand” on page 45.

SAVE/SSAVE Subcommand



Purpose

The SAVE subcommand saves the current session in the dump log file, but does not terminate the VM Dump Tool session.

Operands

dumpfn
is the file name of the dump itself.

newfn
is the new file type to be assigned to the dumplog before saving it.

dumpft
is the new file type to be assigned to the dumplog before saving it.

dumpfm
is the new file mode to be assigned to the dump before saving it.

Usage Notes

1. Without other parameters, the dump log file will be saved as *"dumpname* DUMPLOG A", where *dumpname* is the file name of the dump file. The file name of the dump is displayed at the top of the XEDIT session screen and can be obtained by DUMPNAME subcommand.
2. See also the “FILE/FFILE Subcommand” on page 70 and the “QUIT/QQUIT Subcommand” on page 115.

SCROLL Subcommand

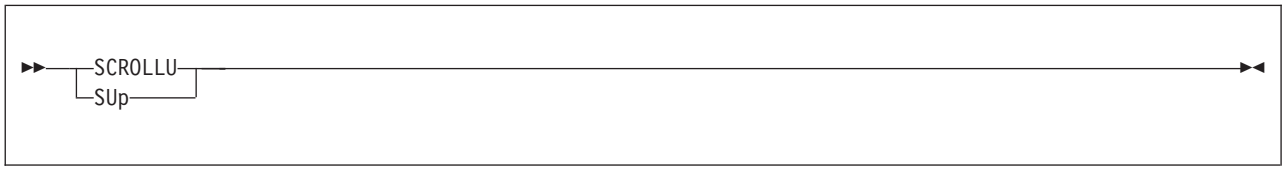
Purpose

The SCROLL subcommand displays the next section of storage following a display of zero length.

Usage Notes

The subcommand is allowed only after a full-screen DISPLAY of zero length. Other (non-DISPLAY) commands are allowed between the DISPLAY and this subcommand.

SCROLLU Subcommand



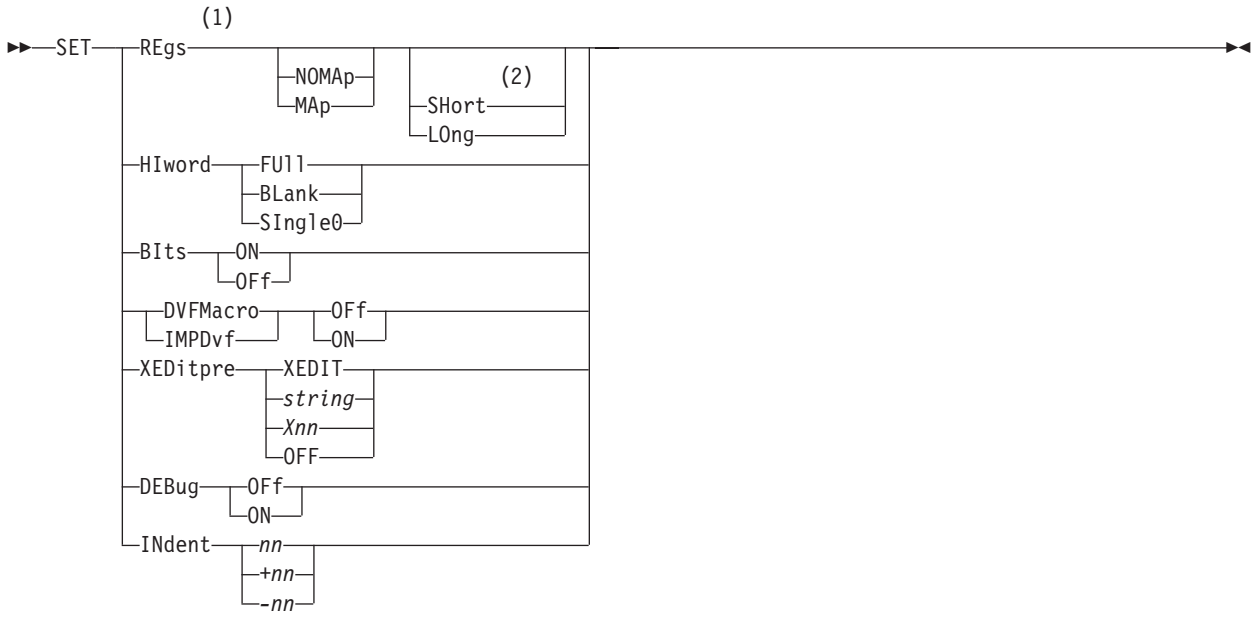
Purpose

The SCROLLU subcommand displays the previous section of storage following a display of zero length.

Usage Notes

The subcommand is allowed only after a full-screen DISPLAY of zero length. Other (non-DISPLAY) commands are allowed between the DISPLAY and this subcommand.

SET Subcommand



Notes:

1 Following SET REGS you must choose at least one operand from the two pairs (MAP/NOMAP or SHORT/LONG). You can choose up to two operands (one from each pair).

2 The default depends upon the size of registers in the dump.

Purpose

The SET subcommand is available to let the user tailor the output and operation of some subcommands and macros.

While the SET subcommand will continue to be supported for compatability with previous releases, the VMDTSET subcommand is preferred. New function will be added to VMDTSET only.

Operands

REgs
specifies how register values are to be displayed from the GREGS subcommand. Options are:

MAP
the value of each register is mapped to a module and displacement.

NOMAP
Register values are not to be mapped to modules.

SHORT
Register values are assumed to be four bytes long.

LONG
Register values are assumed to be eight bytes long.

HIword
specifies how to display double-word register values when the high order full-word of the 64-bit data is zero. Options are:

FULL
the display will consist of the full double-word data with an underscore (_) between the 2 words.

BLANK
the display will consist of only the underscore (_) and the second full-word.

Single0
the display will consist of a single zero, the underscore (_), and the second full-word.

BIts
specifies whether bit definitions are to be displayed automatically with VM Dump Tool subcommands, macros and a number of trace formatters. Options are:

ON bit and code meanings are displayed.

OFf
bit and code meanings are not displayed

DVFMacro
specifies whether ADDRESS SCAN input is accepted or rejected. Options are:

ON specifies that ADDRESS SCAN input will be accepted and processed.

OFf
specifies that ADDRESS SCAN input will be rejected with message HCQ106E.

IMPDvf
specifies whether command line input is used to search for a SCAN macro. Options are:

ON specifies that SCAN macros will be included in the search sequence when an input subcommand is not recognized as a VMDT macro, a VM Dump Tool subcommand, or an XEDIT macro.

OFf
specifies that command line input should not be used to search for a SCAN macro.

XEDitpre
defines an XEDIT prefix string, which allows you to pass to XEDIT a command that would normally be interpreted directly by the VM Dump Tool. Options are:

string
is a one- to eight-character EBCDIC string that the VM Dump Tool recognizes as the XEDIT prefix string. This value cannot be any of the following: FILE, FFile, SAVE, SSave, Quit, QQuit (or abbreviations thereof), OFF, or Xnn where nn is a valid hexadecimal value. The default is the string XEDIT.

X*nn*
is the hexadecimal value of a one-character string that VM Dump Tool recognizes as the XEDIT prefix string.

OFF
specifies that there is no XEDIT prefix string.

DEBug
Specifies whether error messages and non-zero return codes produced from running a VM Dump Tool macro should be displayed on the virtual machine console. Options are:

ON provides a running display on the virtual machine console of all error messages and non-zero return codes produced from running a VM Dump Tool macro.

OFf
specifies that error messages and non-zero return codes produced from running a VM Dump Tool macro are not displayed on the virtual machine console.

INDENT
Sets the offset from the left margin for output. It is usually used only within macros. Options are:

nn is a one- to two-digit decimal number that specifies an absolute indentation of *nn* spaces.

+*nn*
specifies an increased indentation of *nn* spaces.

-*nn*
specifies a decreased indentation of *nn* spaces.

Usage Notes

1. LONG or SHORT defaults are based on dump type. A 32–bit dump defaults to SHORT registers. A 64–bit dump defaults to LONG registers.
2. LONG and SHORT affects GREGS; CPEBK is controlled by the format of CPEBK or LONG, SHORT, or SPLIT options on the CPEBK subcommand.
3. MAP and NOMAP affects GREGS and CPEBK subcommands.
4. Usage notes for XEDITPRE:
   - When you define an XEDIT prefix string, the actual string is stripped off before the command is passed to XEDIT.
   - The VM Dump Tool checks for an XEDIT prefix string before performing any other command handling functions. If the XEDIT prefix string is set to the name of an existing command or macro, the XEDIT prefix string will be recognized from the command line instead of the command or macro. For example, if the XEDIT prefix string is set to *, a line starting with * is no longer recognized as a comment line.
   - If you set the XEDIT prefix string to a single character, you do not have to leave a blank between the XEDIT prefix string and the command being passed to XEDIT. If you set the string to longer than one character, a blank is required between the string and the next term on the input line.
   - You should avoid the use of an alphabetic character as a single-character XEDIT prefix string because the VM Dump Tool will recognize any command or macro beginning with that character as starting with the XEDIT prefix string. For example, if you set the XEDIT prefix string to “s”, you can no longer issue any VM Dump Tool commands or macros which start with “s” (such as the SET subcommand). If the XEDIT prefix string is set to a character that you cannot reset, FILE or QUIT the dump session and restart it.

- The XEDIT prefix string applies only to input entered on the command line. The string is not recognized when the input is from a macro. A macro should use ADDRESS XEDIT to send commands to XEDIT directly.
- Lowercase characters are changed to uppercase before processing.
- When you specify a hexadecimal value (using the Xnn format), the EBCDIC character represented by nn is recognized as the XEDIT prefix string. For example, if you issue SET XEDITPRE X61, the slash (/) character is recognized as the XEDIT prefix string.

5. Corresponding QUERY options are provided for querying the status of SET options.
6. Some subcommands and macros have operands which can temporarily override the BITS or NOBITS setting for the duration of that subcommand or macro.

Examples

Typical output from the SET subcommand:

```
>>> query all
REGS     set to NOMAP    SHORT
HIWORD   set to FULL
BITS     set to ON
INDENT   set to  0
DVFMACRO set to OFF
IMPDVF   set to OFF
MACLVL   set to  0
XEDITPRE set to XEDIT
DEBUG    set to oFF
>>> set bits off
complete

>>> set regs map
complete

>>> set regs long
complete

>>> set dvfmac on
complete

>>> set impdvf on
complete
>>> set debug on
complete

 >>> query all
REGS     set to MAP   LONG
HIWORD   set to FULL
BITS     set to OFF
INDENT   set to  0
DVFMACRO set to ON
IMPDVF   set to ON
MACLVL   set to  0
XEDITPRE set to XEDIT
DEBUG    set to ON


>>> set xeditpre /
complete
```

Sets the XEDIT prefix string to the single character **/**. If you then issue **'/ /123/'** or **'//123/'**, the first **/** is the XEDIT prefix string and is removed. The remainder of the line, `'/123/'` is passed to XEDIT (as the command to locate the string `'123'`).

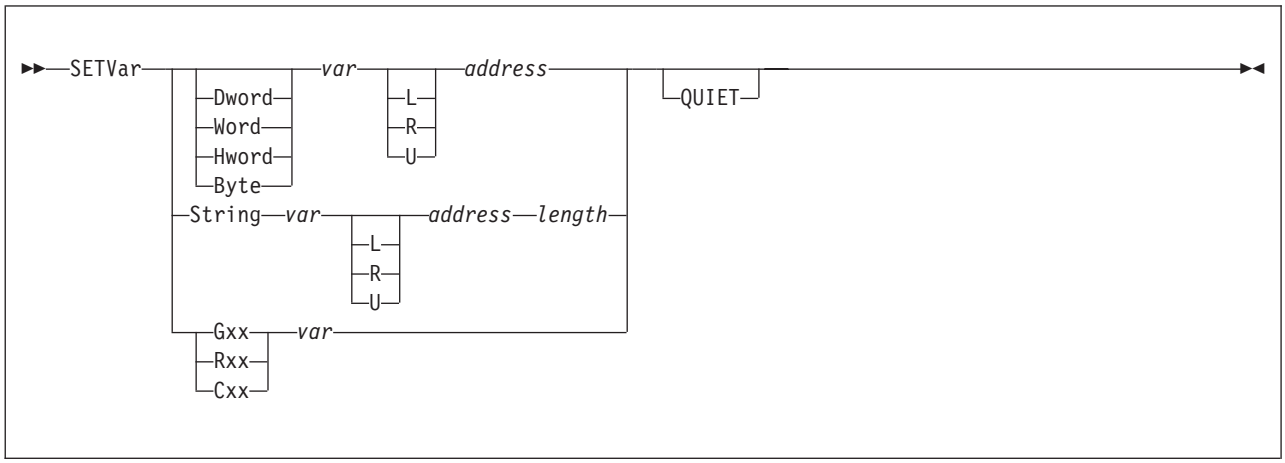
```
>>> set xeditpre quark
complete
```

Sets the XEDIT prefix string to `'quark'`. Because the XEDIT prefix string contains more than one character, you must use a blank between it and the string being passed to XEDIT. For example, if you issue **'quark /123/'**, `'quark'` is recognized as the XEDIT prefix string and is removed. `'/123/'` is passed to XEDIT as in the previous example.

```
>>> set debug on
complete
```

SETVAR Subcommand

PI



Purpose

The SETVAR subcommand allows you to set REXX variables directly from storage locations or registers from the dump. It works only within a VM Dump Tool macro.

Operands

Dword
indicates that the variable is to be loaded with a double-word from the address indicated.

Word
indicates that the variable is to be loaded with a full-word from the address indicated.

Hword
indicates that the variable is to be loaded with a half-word from the address indicated.

Byte
indicates that the variable is to be loaded with a single byte from the address indicated.

G*xx*
R*xx*
indicates that the variable is to be loaded from a general register.

C*xx*
indicates that the variable is to be loaded from a control register.

String
indicates that the variable is to be loaded with a variable length string from the address indicated.

var
is the name of the REXX variable to be set.

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or

U for User-defined. There should be no space between the prefix character and *address*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

address
is the one- to sixteen-significant digit hexadecimal address in the dump from which to obtain the data.

length
is the decimal length of data to be obtained, and is applicable only to STRING type variables. The maximum length allowed is 4096 bytes.

QUIET
indicates that message 'HCQ004E Page not in dump' should be suppressed.

Usage Notes

1. For all types other than STRING, the data is converted to readable hexadecimal before setting the variable. Specification of STRING causes the data to be set in the variable for the length specified without conversion.
2. The register values loaded are always from the failing processor only.
3. See also the SETVAR option on the "BLOCK Macro" on page 30.
4. For 32–bit registers, the variable is loaded with 8 characters of readable hexadecimal.

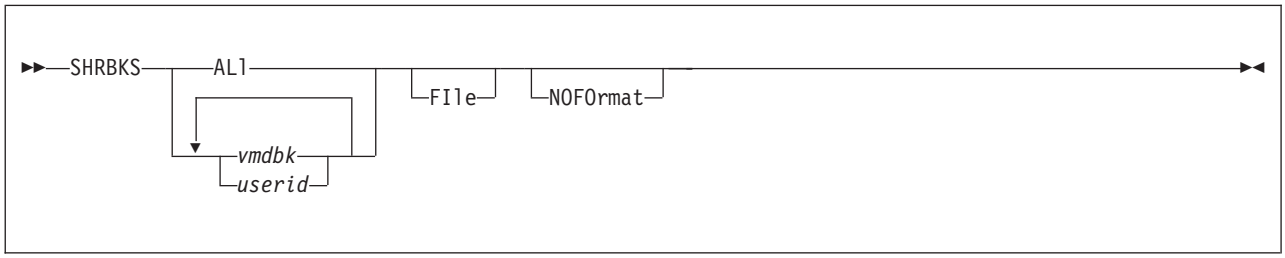
   For 64–bit registers, the variable is loaded with 17 characters. The middle (9th) character is always an underscore (_). The last 8 characters are always the readable hexadecimal of the low order word. If the high order word of the register is not zero, the first 8 characters loaded into the variable are the readable hexadecimal of that value. If the high order word is zero, the first 8 characters loaded into the variable are formatted per the SET HIWORD setting.

Examples

Typical output from the SETVAR subcommand:

```
>>> display 0.10
00000000_00000000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*

>>> setvar word xxx 0

(xxx has been set to 00FC3000)

>>> setvar hword yyy 4

(yyy has been set to 8467)

PI end
```

SHRBKS Macro



Purpose

The SHRBKS macro is used to list the SHRBKs for one of, a list of, or all of the VMDBKs in the system.

Operands

ALL
specifies that all VMDBKs in the global cyclic list are to be analyzed. This may generate a large amount of output, so you may want to consider using the FILE option to separate this data from the current dump session. Also, you can stop processing by using HI.

vmdbk
is the one- to eight- significant digit hexadecimal logical address in the dump of a VMDBK for which the SHRBKs are to be found and displayed. Any number of VMDBKs or user IDs may be specified. VMDBKs and user IDs may be intermixed in any order.

userid
is the one- to eight-character user ID that displays and analyzes the SHRBKs for the VMDBK(s) of the user(s) indicated. If a user ID is specified that is the same as a VMDBK address, this will be treated as an address. To handle such a user ID, you must specify the address of the VMDBK for that user. Any number of user IDs may be specified. VMDBKs and user IDs may be intermixed in any order.

FIle
causes the output to be placed in a file called "dumpname SHRBKS A1". If the file already exists, the new information will be appended to the existing file.

NOFOrmat
generates just the data lines without the header lines before the data and the blank line after the data.

Usage Notes

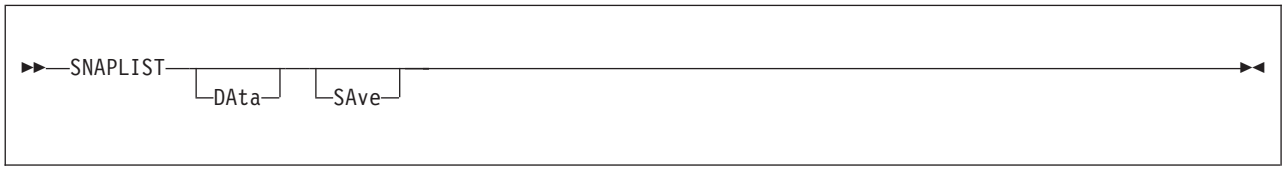
1. You may specify a single parameter as an underscore character ("_"). Place the cursor on a blank-delimited term in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value at the cursor.
2. The NOFOrmat option is useful when invoked from a macro.
3. If a user ID is specified, only the base VMDBK for that user is used.
4. If a user ID is also a VMDBK address, it will be interpreted as a VMDBK address.
5. This macro is supported only for CP dumps.

Examples

Typical output from the SHRBKS macro:

```
>>> shrbks 017BB000 tcpip
                                                    SPACE    SPACE
    VMDUSER  VMDBK@   SHRNAME  SHRBK@   SHRSNTPT NAME     SNTBK@
    -------- -------- -------- -------- -------- -------- --------
    OPERATOR 017BB000 INSTSEG  01D03010 01D25000
    TCPIP    01E11000 NLSAMENG 01E228C0 01E10000
    TCPIP    01E11000 CMSVMLIB 01E22900 01D60000
    TCPIP    01E11000 CMSPIPES 01E22940 01D49000
    TCPIP    01E11000 INSTSEG  01E22A60 01D25000
    TCPIP    01E11000 CMS      01E22AC8 01D93000
2 of 2 VMDBKs processed, 6 SHRBKs found

>>> shrbks all
                                                    SPACE    SPACE
    VMDUSER  VMDBK@   SHRNAME  SHRBK@   SHRSNTPT NAME     SNTBK@
    -------- -------- -------- -------- -------- -------- --------
    AUTOLOG1 01D15000 CMSVMLIB 01D38B08 01D60000
    AUTOLOG1 01D15000 CMSPIPES 01D38B88 01D49000
    AUTOLOG1 01D15000 INSTSEG  01D19038 01D25000
    DISKACNT 01D08000 CMSVMLIB 01D38B48 01D60000
    DISKACNT 01D08000 CMSPIPES 01D38BC8 01D49000
    DISKACNT 01D08000 INSTSEG  01D18008 01D25000
    EREP     01D09000 CMSVMLIB 01D38AC8 01D60000
    EREP     01D09000 CMSPIPES 01D38C08 01D49000
    EREP     01D09000 INSTSEG  01D18048 01D25000
    ERNSBERW 01D16000 CMSVMLIB 017B04A0 01D60000
    ERNSBERW 01D16000 CMSPIPES 017B04E0 01D49000
    ERNSBERW 01D16000 INSTSEG  017B05E8 01D25000
    ERNSBERW 01D16000 CMS      017B0658 01D93000
    GCSXA    01D87000 VTAMXA   01D89A60 01D94000
    GCSXA    01D87000 GCSXA    01D89AC8 01D86000
    OPERATOR 017BB000 INSTSEG  01D03010 01D25000
    OPERSYMP 01D17000 CMSVMLIB 01D38A88 01D60000
    OPERSYMP 01D17000 CMSPIPES 01D38C48 01D49000
    OPERSYMP 01D17000 INSTSEG  017C9088 01D25000
    PVM      01DED000 INSTSEG  01DEF950 01D25000
    RSCS     01DA7000 VTAMXA   01DB0CB0 01D94000
    RSCS     01DA7000 GCSXA    01DB0D18 01D86000
    RSCSDNS  01DA4000 SCEEX    01DB05D0 01DDD000
    RSCSDNS  01DA4000 CMSVMLIB 01DB0650 01D60000
    RSCSDNS  01DA4000 CMSPIPES 01DB0690 01D49000
    RSCSDNS  01DA4000 INSTSEG  01DB0A48 01D25000
    RSCSDNS  01DA4000 CMS      01DB0BF8 01D93000
    SYSTEM   00002000 No SHRBKs found
    TCPIP    01E11000 NLSAMENG 01E228C0 01E10000
    TCPIP    01E11000 CMSVMLIB 01E22900 01D60000
    TCPIP    01E11000 CMSPIPES 01E22940 01D49000
    TCPIP    01E11000 INSTSEG  01E22A60 01D25000
    TCPIP    01E11000 CMS      01E22AC8 01D93000
12 of 12 VMDBKs processed, 32 SHRBKs found
```

SNAPLIST Macro



Purpose

The SNAPLIST macro displays the addresses, data and/or CPEBKs of the Snap Data Parameter List associated with a soft abend dump.

Operands

DAta
specifies that the full data areas are to be displayed.

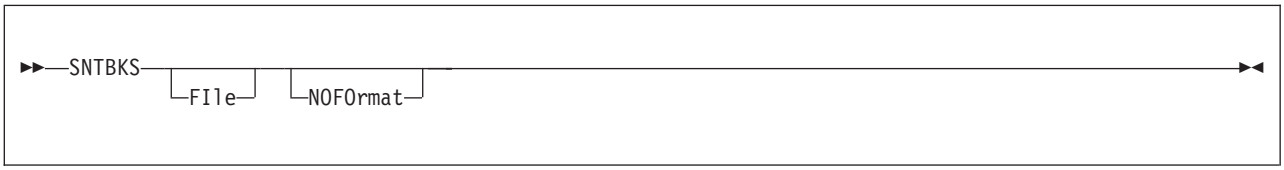
SAve
specifies that each CPEBK in the list be formatted and displayed.

Examples

Typical output from the SNAPLIST macro:

```
>>> snaplist

Snapped Data Areas

Snapped  --Preserved--
Address  Lng  Address  Tag
007972C0 00E0 01F69EC0 ASCBK
007972C5 0028 01F42098 SIDBK
007972CA 01D8 01F423B8 ZLCBK

Snapped Save Areas

Snapped  Preserved
Address  Address
01FDB780 00046370
01FBD200 000463F0
```

SNTBKS Macro



Purpose

The SNTBKS macro is used to list the SNTBKs on the following queues:

- Saved segments
- Named Saved segments
- SAVESEG segments
- SAVESYS segments.

Output includes the name of the queue where it was found, the name of the SNT, the address of the SNTBK, flags from that block, the number of users who are currently using this SNTBK, the number of users who have Exclusive access to this SNTBK, the address of the SFDBK and STLBK associated with this SNTBK, and the address of the associated SHRBK queue.

Operands

FIle
causes the output to be placed in a file called 'dumpname SNTBKS A1'. If the file already exists, the new information is appended to the existing file.

NOFOrmat
generates just the data lines without the header lines before the data and the blank line after the data.

Usage Notes

This macro is supported only for CP dumps.

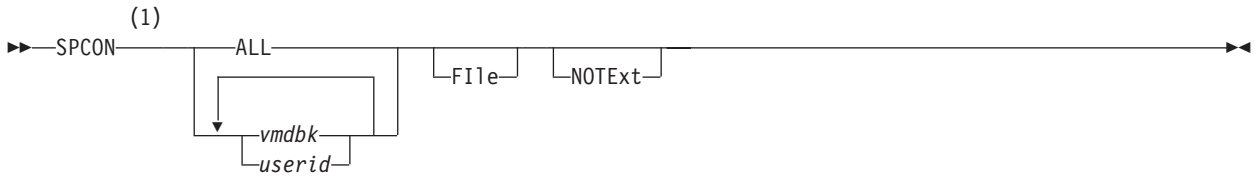
Examples

Typical output from the SNTBKS macro:

```
>>> sntbks
QUEUE SNTNAME     SNTBK@   FLAG SHR# XCL# SDFBK@   STLBK@   SHRBK Q@
----- ----------- -------- ---- ---- ---- -------- -------- --------
NSUNSGAN   (Saved Segments)
      TSTSP2      04222000 2100    0    0 772A61B0 772A6148 00000000
         TSTDCS1  345EC000 1100    1    0 4989C700 00000000 3DE65A50
         TSTDCS3  349F8000 1100    2    0 773DDDC8 00000000 3DE65888
      LE16SCEE    3ABA4000 0100    4    0 3ABA5F60 3ABA5E78 385A33D8
      DFSMSSEG    55AD3000 8100    8    0 76D40F60 4D918498 46CBF008
      C13SEG      585DC000 0100    B    0 5A8463E8 5A846648 3B0AE4E8
      C13MISC     5A9FC000 0100    B    0 75FBCF60 72F10008 765173E0
      C14SEG      66832000 0100    A    0 66AE9A20 66AE9950 44ECD930
      CSL130      6C640000 0100    A    0 76D41F60 6C63FE80 3DAC0F48
      C14MISC     6E427000 0100    A    0 76D420B0 7152D2D8 73B5B4A8
      VTAM        7395E000 0100    4    0 72F10598 72F10470 3B69E728
      DOSBAM      732A4000 2100    0    0 73AD9028 73090D40 00000000
         CMSBAM   7310F000 1100    4    0 74A3D2B0 00000000 3B69E768
         CMSVSAM  734EC000 1100    4    0 73AD9190 00000000 3B69E7A8
```

```
         CMSAMS   730E1000 1100    0    0 74A3D0E0 00000000 00000000
         CMSDOS   7308D000 1100    0    0 73090E70 00000000 00000000
      C16SEG      75607000 0100   42    0 75ACEB90 75ACEAA8 4A697BD8
      C12SEG      757E2000 0100    2    0 75ACEE48 75ACED60 479DFDF8
      C12MISC     75C9A000 0100    2    0 7666E268 765B5490 3DAC0F88
      C15MISC     75CA4000 0100   32    0 765B54E0 765B54B8 3D801FC8
NSUNSYAN   (Named Saved Segments)
      GCSRSCS     3DCDD000 0200    1    0 3DE61D48 3DE61ED8 3DE625A8
      C13         63CCB000 0200    B    0 64FBD260 64FBD178 76517420
      CMS130      73ABF000 0200    A    0 73AD95F0 73AD9508 71279D28
      C14         74021000 0200    A    0 74A3DA00 74A3D918 4A697C18
      C15         74667000 0200   10    0 74D49F30 74D49878 44EDAA00
      C16         74815000 0200    9    0 74B93BF8 74DD4130 479DF080
      GCSVTAM     74D10000 0200    4    0 74D30098 74CA48F0 3B69E7E8
      C12         764F5000 0200    2    0 76518728 76518640 7651B078
      CMS         76522000 0200   2A    0 7651DF60 76518928 3B0611E0
NSUSSGAN   (DEFSEG/SAVESEG Command)
      No SNTBKs on queue
NSUSSYAN   (SAVESYS Command)
      No SNTBKs on queue
29 SNTBKs processed
```

SPCON Macro



Notes:

1 Keywords, user IDs, and VMDBK addresses may be intermixed in any order. Anything not recognized as a valid keyword will be tested as a a VMDBK address, and then a user ID. To dump the information about a user whose user ID is a valid VMDBK address or the same as a keyword recognized by SPCON, use the VMDBK address of the user instead of the user ID.

Purpose

The SPCON macro displays spooled console device buffers from the dump for one or more users. This may be used to find messages that were generated just before the dump was taken.

Messages are formatted according to the CCWs stored with them. They displayed approximately the way they looked to the receiving user in chronological order by user, with the oldest message first.

Individual user IDs or VMDBK addresses may be specified in a list, or the ALL keyword may be used to dump all buffers found.

Operands

vmdbk
specifies a one- to eight-significant digit hexadecimal logical VMDBK address for which analysis is to be done. VMDBKs and user IDs may be intermixed.

userid
specifies a user ID of a VMDBK for which analysis is to be done. VMDBKs and user IDs may be intermixed.

ALL
specifies that all of the VMDBKs found in the global cyclic list should be analyzed for spooled console output. ALL may not be used if user IDs or VMDBKs are specified. Please note that ALL may generate a large amount of output, therefore, you may want to consider using the FILE option.

FIle
causes the output to be placed in a file called "*dumpname* SPCON A1".

NOTExt
specifies that the macro should analyze the VMDBKs indicated and include the summary information in the output, but the text of the messages themselves should be suppressed.

Usage Notes

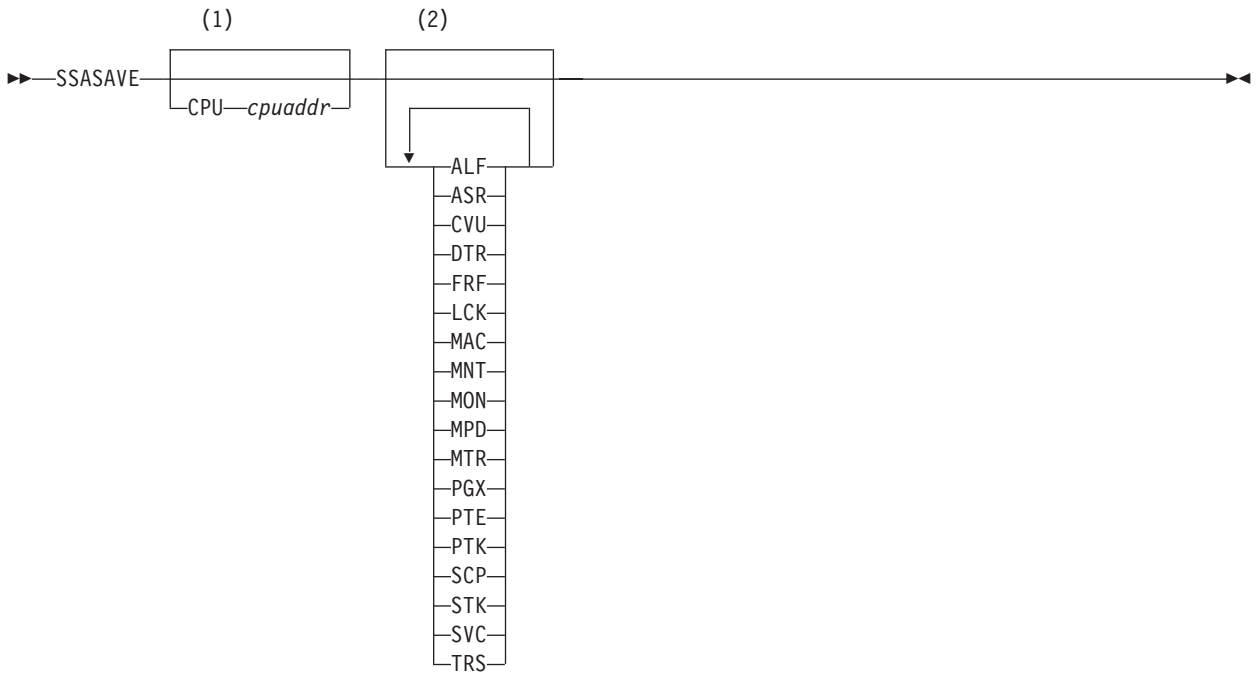
1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. This macro is supported only for CP dumps.

Examples

Typical use of and output from the SPCON macro:

```
>>> spcon tomdef

User TOMDEF    VMD 094DC000  has no spooled output
1 VMDBK of 1 processed
>>> spcon briankt
User BRIANKT   VMD 23FE2000  Dev 0009  VDEV 10B66228  Page 0000000021CE5
   11:53:36  * MSG FROM BATCH71 : TBATCH Starting in BATCH71 at 11:53:36
   11:53:48  * MSG FROM BATCH71 : TBATCH Completed. Sending A-Disk files
   Job BR125560 has ended; RC = 0
   Ready; T=0.12/0.15 11:54:38
   pl
1 VMDBK of 1 processed
```

SSASAVE Macro



Notes:

1 The default is the abending processor.

2 The default is to dump all areas.

Purpose

The SSASAVE macro displays one or more Local Processor Static Saveareas for a selected processor.

Operands

CPU *cpuaddr*
is a one- to four-digit hexadecimal CPU address for which the saveareas are to be displayed. The default is the abending processor.

ALF
Requests selection of the SSAALF savearea (RSM related).

ASR
Requests selection of the SSAASR savearea (assert related).

CVU
Requests selection of the SSACVU savearea (HCPCVUMD related).

DTR
Requests selection of the SSADTR savearea (trace related).

FRF
Requests selection of the SSAFRF savearea (RSM related).

LCK
Requests selection of the SSALCK savearea (lock related).

MAC
Requests selection of the SSAMAC savearea (RSM related).

MNT
Requests selection of the SSAMNT savearea (monitor related).

MON
Requests selection of the SSAMON savearea (monitor related).

MPD
Requests selection of the SSAMPD savearea (RCPU related).

MTR
Requests selection of the SSAMTR savearea (monitor related).

PGX
Requests selection of the SSAPGX savearea (HCPPGXXE related).

PTE
Requests selection of the SSAPTE savearea (RSM related).

PTK
Requests selection of the SSAPTK savearea (RSM related).

SCP
Requests selection of the SSACP savearea (stack related).

STK
Requests selection of the SSASTK savearea (stack related).

SVC
Requests selection of the SSASVC savearea (trace related).

TRS
Requests selection of the SSATRS savearea (trace related).

Usage Notes

1. If one or more saveareas are selected by name, only those selected will be displayed. If none are selected, then all saveareas will be displayed.
2. You may specify *cpuaddr* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the macro call will be replaced by the value pointed to by the cursor.
3. This macro is supported only for CP dumps.

Examples

Typical use of and output from the SSASAVE macro:

```
>>> ssasave frf
   SSAFRF at 04520080   CPU 0000   PFXPG 04508000 (failing)
FRFFPNT 00000000 FRFBPNT 00000000 FRFSFQP 00000000 FRFCPRQ 00000000
FRF+10= 00000000  FRF+14= 00000000
R0-7 845A9D78 0465FA08 33DEB000 00000235 0000000A FFFFFFFF 00000000 77BBE180
R8-F 845AA000 77BBE180 04500000 04500000 045A9700 77274A00 845AA0BC 045AB5E0
FRFR14 is HCPFRD+9BC
FRFR15 is HCPFRFGF
FRFWRK0-4 00000000 00000000 00000000 00000000 00000000 *.....................*
FRFWRK5-9 00000000 00000000 00000000 00000000 00000000 *.....................*
```

SVGBK Subcommand

SVGBK is a synonym for CPEBK. See “CPEBK Subcommand” on page 45.

SXSTE Macro

Notes:

1 If neither BITS nor NOBITS is specified, the default is taken from the current VMDTSET BITS setting.

Purpose

The SXSTE macro displays information from the System eXectution Space Table Entry for a given address. In addition, it may be used to display translation results and information from the corresponding entry in the Frame Table.

Operands

address
is the one- to eight-significant digit hexadecimal logical address in the dump for which the SXSTE data is to be displayed.

ALL
DETails
indicates that all of the information should be displayed as if the TRANS and FRMTE options were included.

FRAme
FRMte
indicates that information from the FRaMe Table Entry for the corresponding real page be displayed if it is available.

TRAns
indicates that information about the translation from the input address to a real address should be displayed.

BITs
indicates that definitions for bits involved should be displayed.

NOBits
indicates that definitions for bits involved should not be displayed.

Examples

Typical use of and output from the SXSTE macro.

```
>>> sxste 0
   SXSTE for L00000000_00000000 is at L00000200_00000000
   SXSFPNTG 00000000_5CC3D75C    *....*CP**
   SXSBPNTG 00000000_00000000    *........*
```

```
SXSPTEG  00000000_00000000    *........*
SXSSTATE 00000000_010C0000    *........*

>>> sxste 0 bits
SXSTE for L00000000_00000000 is at L00000200_00000000
SXSFPNTG 00000000_5CC3D75C    *....*CP**
SXSBPNTG 00000000_00000000    *........*
SXSPTEG  00000000_00000000    *........*
SXSSTATE 00000000_010C0000    *........*
  Bits defined in SXSCSB0           (01)
    01 FRAME IN USE BY CONTROL PROGRAM
    01 Page in use by control program
  Bits defined in SXSCSB1           (0C)
    0C (undefined)
  Bits defined in SXSCSB2           (00)
  Bits defined in SXSCSB3           (00)

>>> sxste 0 all
SXSTE for L00000000_00000000 is at L00000200_00000000
SXSFPNTG 00000000_5CC3D75C    *....*CP**
SXSBPNTG 00000000_00000000    *........*
SXSPTEG  00000000_00000000    *........*
SXSSTATE 00000000_010C0000    *........*

Analysis of  00000000_00000000 translated on ASCE 00000000_0000A007
Table type Table address     + displ  contains
---------- ----------------- -------- -----------------
Region 1st (not applicable)
Region 2nd (not applicable)
Region 3rd 00000000_0000A000 00000000 00000000_BFFF4007
Segment    00000000_BFFF4000 00000000 00000000_BD3E1000
Page       00000000_BD3E1000 00000000 00000000_00000000
           (translation was successful)

Frame table for page 00000000_00000000 is at 00000000_80000000
FRMFPNTG  00000000_5CC3D75C    *....*CP**
FRMBPNTG  00000000_00000000    *........*
FRMPTEG   00000200_00000000    *........*
FRMSTATEG 00000000_01040000    *........*
FRMCSB0 01 FRMCSB1 04 FRMCSB2 00 FRMCSB3 00
```

SYMPTOM Macro

Purpose

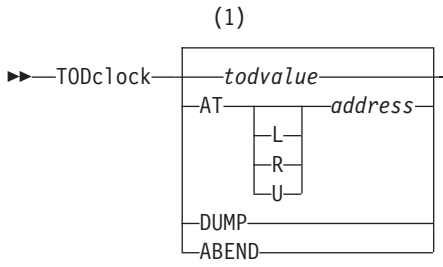
The SYMPTOM macro displays a summary of the symptom record in the dump.

Examples

Typical use of and output from the SYMPTOM macro:

```
>>> symptom
Symptom Record for Incident BCBBA564 CE6CESYM

TOD Clock . . BCBBA564CE6CE1AE         Date. . . . . 03/19/05
Time Zone . . -05.00.00                Time. . . . . 07:58:36.053198
CPU model . . 2064                     Base SCP. . . 5741
CPU Serial. . 031542                   NodeID. . . . GDLSPRF3
Dump Name . . 2T0P095 DUMP0001 P1      Dump Type . . CPDUMP
Comp ID . . . 5741A05                  Ver/Rel/Mod . V05R02M0
Dump format . 32-BIT
-------------------------------------------------------------
Primary Symptom Strings
              PIDS/5741A0502           (Component ID)
              AB/SSVC002               (Abend Code)
              REGS/FFFFF               (Register/PSW Info)
-------------------------------------------------------------
Section 5 Data:
              USERID DUMPED: SYSTEM
              DUMP RECEIVER: OPERATNS
              SPOOLID: 0002
-------------------------------------------------------------
Last trace entry on abending processor
0F280140 07:58:36 Call xR>LR from HCPPTE+F4F to HCPALFMF sav 0177B600
                  parm 00000002
-------------------------------------------------------------
Abend  Description
SVC002 (Hard Abend) A restart interrupt occurred.  For a first level
       system, a restart interrupt occurs when the primary system
       operator selects the restart function on the hardware console.
       For a second level system, a restart interrupt occurs when the
       "SYSTEM RESTART" command is entered on the first level console.
```

TODCLOCK Subcommand



Notes:

1 The default is the current time.

Purpose

The TODCLOCK subcommand displays tod clock values as date and time. The TOD clock values can be supplied or can come from the dump.

Operands

todvalue
is a string of eight or sixteen bytes of hexadecimal data. The data is converted as if it were the value in the TOD clock. If only eight bytes are supplied, zeroes are added to the right hand end. If no parameters are entered, then the current time of day and date are displayed.

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *address*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

AT *address*
is the one-to-eight significant digit hexadecimal address of an eight-byte area that contains the TOD value to be displayed.

DUMP
displays the time and date dump was taken.

ABEND
displays the time and date dump was taken.

Usage Notes

1. The output of the TODCLOCK subcommand is formatted according to the current time zone and DATEFORMAT setting.
2. Some clock times are captured and converted when the VM Dump Tool is initialized. If the DATEFORMAT setting is changed after initialization, those times converted during initialization will not reflect the new setting of SET DATEFORMAT. An example is the CODE subcommand. The time the dump was taken is extracted during initialization. Thus, the output of the CODE subcommand will reflect the SET DATEFORMAT value at entry.

3. Subcommands and macros that invoke the TODCLOCK subcommand dynamically will find that the output of TODCLOCK will be formatted according to the current DATEFORMAT setting. An example is the TODCLOCK subcommand with no operands. The current time is obtained and converted, then displayed according to the current DATEFORMAT setting.

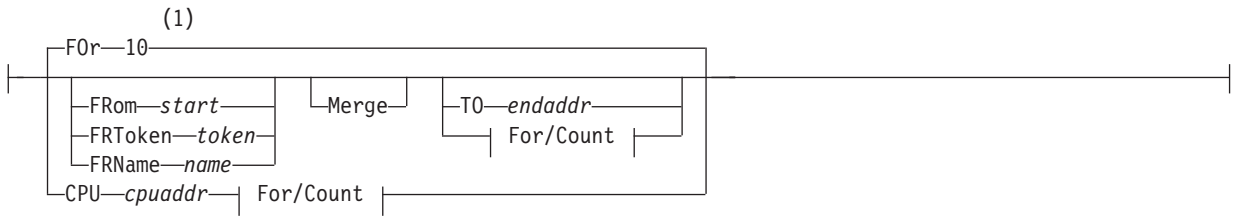
Examples

Typical use of and output from the TODCLOCK subcommand:

```
>>>> tod at E000
Date 03/19/05 Time 07:55:33.136367

>>>> tod dump
Date 03/19/05 Time 07:58:36.053198
```

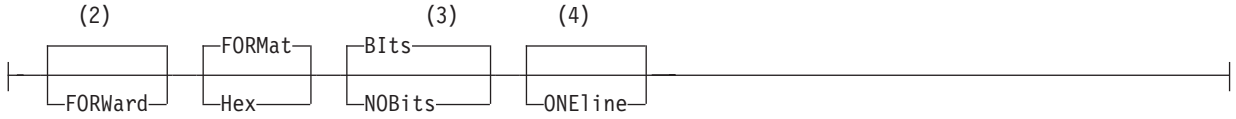
TRACE Subcommand

Range:

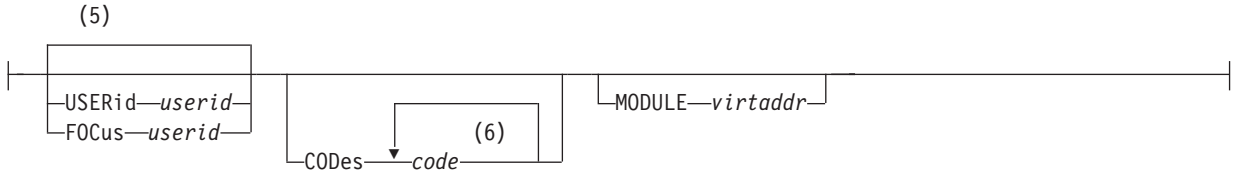


For/Count:

Options:



Filter1:



Notes:

1 The default is the failing CPU only, not merged, backward in time from the most recent trace table entry. The default count is 10 or the count used on the most recent previous TRACE subcommand.

2 The default is to work backward, from the most recent trace entry.

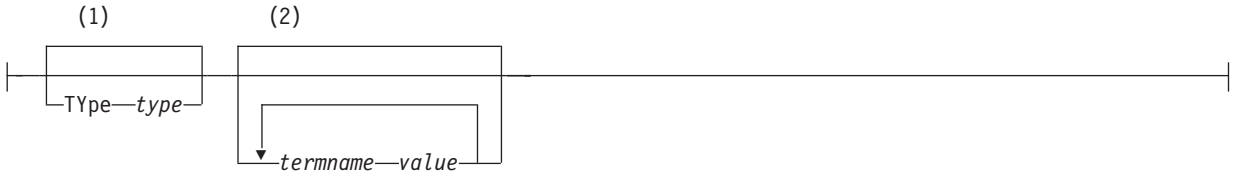
3 The default for bits is taken from the VMDTSET BITS setting.

4 The default for oneline is to display the data into multiple lines.

5 The default is to include all users without providing FOCUS details on any.

6 You may specify up to 100 four-digit hex trace codes to be included.

Filter2:



Notes:

1 The default is to not filter by type. The list of types is variable and may be redefined by the user. Issue VMDTQRY TRACE TYPE to get a list of the types available for you to specify.

2 The default is to not filter by terms. The list of terms is variable and may be redefined by the user. Issue VMDTQRY TRACE TERM to get a list of the types available for you to specify.

Purpose

The TRACE subcommand displays formatted entries from the trace tables of one or more processors within the dump.

Operands

BIts
specifies that bit definitions are to be displayed. BITS cannot be specified with NOBITS.

CODes *code*
selects specific types of trace entries by their four-digit hexadecimal trace codes. The CODES keyword can be followed by a list of up to 100 four-digit hexadecimal trace codes. This option is useful when there is no predefined list (see the TYPE option) that satisfies your requirement.

For example, you can locate CP commands (which have a trace entry code of 2301) by "TRACE CODES 2301".

The codes selected with this option will be used in addition to any selected with the TYPE option.

CPU *cpuaddr*
is a one- to four-digit hexadecimal CPU address whose trace table is to be processed. The default is the abending processor. CPU cannot be specified with MERGE, FROM, or TO.

MODULE *virtaddr*
specifies either

- a module name or an entry point name optionally followed by a plus sign (+) and a one- to eight-significant digit hexadecimal displacement (example: HCPxxx+200), or
- a one- to eight-significant digit hexadecimal CP logical address.

This value is used to limit the trace entries processed to only calls to or returns from the specified value.

If the dump is from a release prior to z/VM version 5, release 1.0, and the desired entry point is pageable, you must specify the host virtual address rather than the name. Use the MAP and VIRTUAL commands to get that address.

FOCus *userid*
specifies the one- to eight-character user ID for which suffixes are displayed with no filtering.

For a user ID with multiple VMDBKs, each time the user ID is displayed, it is suffixed as follows:
userid.**BASE**
- primary VMDBK for this user

userid.*nn*
- secondary VMDBK for cpu *nn*

userid.VSIE.*nn*
- secondary VSIE VMDBK for cpu *nn*

If there is just one VMDBK for this user ID, no suffix is added.

FOr *num*
Count *num*
specifies the decimal number of trace entries to display. Using an asterisk (*) instead of a number specifies that all entries will be displayed until the end of the trace table is reached. If this option is not specified the initial default is 10. After each TRACE subcommand, the number is saved, and will be used as the default for the next TRACE subcommand. FOR or COUNT cannot be specified with TO. A *num* value of 0 is treated as *.

FORMat
specifies that the trace entries be formatted before being displayed. FORMat is the opposite of Hex.

FORWard
specifies that trace entries be shown in ascending time sequence. This option also modifies the default start and end points of the TRACE command. Normally the default start entry is the most recent, and the default stop entry is the oldest. This option reverses the start and stop default entries. Thus "TRACE FORWARD FOR 1 MERGE" will display the oldest trace entry in the table.

FRom *start*
specifies the one- to eight-significant digit hexadecimal real address of the starting trace entry to be displayed. FROM cannot be specified with CPU.

FRToken *token*
specifies the token of the Function Related trace table to be formatted.

FRName *name*
specifies the name of the Function Related trace table to be formatted.

Hex
specifies that the trace entries will be displayed in hexadecimal, rather than formatted into readable English. The TOD clock value from the trace entry and the CPU will optionally be formatted depending on other options (see the VMDTSET TRACE command). If you want to see the entire trace entry in hex, use the DISPLAY command.

Merge
displays the trace tables for all CPUs in the dump, merged according to the TOD clock values in the trace entries. Merge cannot be specified with CPU.

NOBits
specifies that no analysis of bits in bytes will be displayed with the trace entries. This can reduce the size of the output considerably, especially for entries with many bytes to be analyzed, such as Test Subchannel trace entries. NOBITS cannot be specified with BITS.

ONEline
specifies that only one line of output will be displayed for each trace entry. This displays the most significant part of the information from the entry, and considerably reduces the amount of output from the TRACE command. One implies NOBITS.

termname value
specifies the name of a term to be searched for, and the value of that term that must match for the trace entry to be displayed.

When one *termname* is specified, any trace entry that contains that term and matches the value specified for that term will be displayed.

When multiple *termname*s are specified, a trace entry contains only one of the terms specified, and it matches the value specified for that term, the trace entry will be displayed. If the trace entry contains more than one of the *termname*s specified, then all of the values in the trace entry must match those specified for the terms specified for the trace entry to be displayed.

If a trace entry includes none of the specified *termname*s, then it will not be displayed.

The list of *termname*s available is variable and may be changed by the user (see Appendix G, "Using the Trace Format Definition Table," on page 237). The list of *termname*s currently available can be obtained with VMDTQRY TRACE TERM.

The list that follows is an example of *termname*s that may be available.

```
ADDRESS  ASCBK    ASTE     CARBK    CCTBK    COMBK
CPEBK    DUMPCODE ESW      EXTCODE  FCTBK    FRMTE
HASHID   IACCODE  IORBK    IPARML   IUCVB    IXBK
IXBLK    LATBK    LDEVNO   LNKBK    LOCBK    LOCTHRED
LSCH     MDEBK    MODULE   MSGBK    PATH     PGMCODE
PTHBK    RDEV     RDEVNO   RSCH     SID      SNABK
SUBCH    SVCCODE  SYSSVCCD TCHBK    TCHKEY   TCHTKFMT
TSKBK    VDEV     VDEVNO   VMDBK    VSCH     WEIBK
XCRBK    XITBK
```

In addition, if you want to search for values that you know are in a given trace entry type, you can do a VMDTQRY TRACE ENTRY nnnn to display the terms that are included in that trace entry.

TO *endaddr*
specifies the one- to eight-significant digit hexadecimal real address of the last trace entry to be displayed. TO cannot be specified with FOR, COUNT, or CPU.

TYpe *typenames*
specifies types of trace entries to be displayed.

The list of types available is variable and may be changed by the user (see Appendix G, "Using the Trace Format Definition Table," on page 237). The list of *typenames* currently available can be obtained with VMDTQRY TRACE TYPE.

The list that follows is an example of *type*s that may be available.

```
APPC     CALL     CCALL    CCS      CONTROL  EXIT
FREE     I/O      IO       IUCV     LI/O     LIO
LOCK     MDC      PC       RI/O     RIO      SCSI
SENSE    SIE      STACK    STORAGE  VCTC     VI/O
VIO      VSTORAGE
```

USERid *userid*
specifies the one- to eight-character user ID of interest. Any trace entry which contains any VMDBK for this user ID will be shown. If other filtering options are also specified, a trace entry is displayed only if meets all of the criteria specified.

The user ID output is displayed with suffixes as documented above with the FOCUS operand.

A suffixed user ID can not be specified with the USERid operand. Use the VMDBK address with the VMDBK operand instead.

Usage Notes

1. In general, you can use as many of these options as you want on the TRACE command to limit the trace entries that will be shown. You must avoid certain combinations that are not supported as described above. Note that the command length is limited to 512 characters.
2. The FROM and TO addresses are not checked for actually being in the trace table. If the addresses specified are not part of the trace table, unpredictable results can occur.
3. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the data area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
4. To halt a long-running TRACE command, press the PA1 key to break out of full screen mode and then enter the HI command.
5. For Function Related Trace Tables:
   - FROM and TO addresses must be prefixed with an L to indicate they are logical addresses.
   - If a FROM or TO address is prefixed with an L, that address will be used as an implicit reference to the Function Related Table being requested (no token or name for the table will be required).
   - If the FROM and TO addresses do not refer to the same Function Related Trace Table, then data from the table will be displayed from the FROM address to the end of the table.
   - No function to merge any combination of Function Related trace tables and main trace tables is provided.
   - Note that tokens and names of the Function Related trace tables in the dump may be found using the VMDTQRY subcommand.
6. This subcommand is supported only for CP dumps.

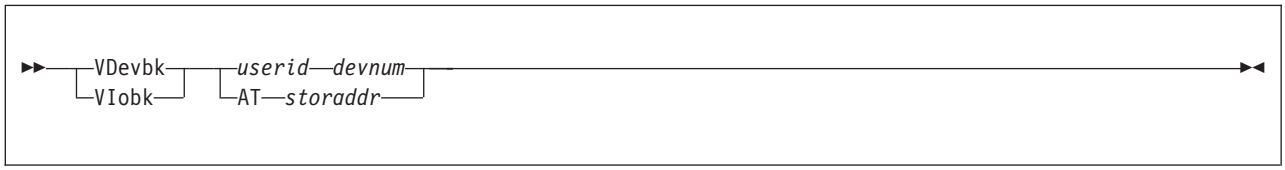
Examples

Typical use of and output from the TRACE subcommand is as follows:

```
>>> trace for 10 merge one
7D9E1D60 CPU 0001 Emerg Signal ExtInt from cpu 000B parm 0001D000
7D9E1D40 CPU 0001 Exit to dispatcher fr HCPDSW+550 vmdbk 0432F000
7D9E1D00 CPU 0001 Monitor Call at HCPDSP+1E0 instr MC     4,X'F'
7D9E1CC0 CPU 0001 Monitor Call at HCPASR+1C16
7D9E1C80 CPU 0001 Monitor Call at HCPASR+1AAC
```

```
7D9E1C40 CPU 0001 Monitor Call at HCPDSW+BA instr MC     X'30'(R12),X'F'
7D9E1C20 CPU 0001 Rtrn to HCPRUN+D38 fr HCPTMR+47E cpebk 075AAA00
7D9E1C00 CPU 0001 Call fr HCPRUN+D38 to HCPTMREW cpebk 075AAA00
7F780000 CPU 0000 Emerg Signal ExtInt from cpu 000B parm 0001D000
7D162C40 CPU 0002 Emerg Signal ExtInt from cpu 000B parm 0001D000

>>> trace for 5 cpu 3 one
7D1B3880 20:33:44  Emerg Signal ExtInt from cpu 000B parm 00000000
7D1B3860 20:33:44  Enter-wait-state wait-mask 369E0000_00000000
7D1B3840 20:33:44  Exit to dispatcher fr HCPDSW+560 vmdbk 18C53000 DGA44
7D1B3800 20:33:44  Monitor Call at HCPDSP+1E0 instr MC     4,X'F'
7D1B37C0 20:33:44  Monitor Call at HCPASR+1C16

>>> trace for 5 type io one
7D16B060 20:33:44  SSCH cc0 rdevno 2A6E rsch 1531 iorbk 02B56408
7D95BE20 20:33:44  vTSCH vdevno 0748 rdevno 0748
7D95BAA0 20:33:44  v370xa IoInt vdevno 0748 rdevno 0748 rdevsid 0001
7DD8B860 20:33:44  SSCH cc0 rdevno 2A87 rsch 154A iorbk 026C8E08
7DD8B840 20:33:44  TSCH cc0 rdevno 2A87 rsch 154A esw 00400005

>>> trace for 5 type iucv merge one
7DFB10C0 CPU 0001 IUCV-Int vmdbk 0432F000 TCPIPU00 iucvb 211386C8
7D8CBDE0 CPU 000B IUCV-Rply at 80E150E8 sys-svc (none) msgbk 062FF808
7DFC1720 CPU 0001 IUCV-Send at 8131F066 sys-svc (none) iucvb 211386C8
7D8EBE40 CPU 000B IUCV-Int vmdbk 0631E000 TCPIP iucvb 0633C230
7D8EB940 CPU 000B IUCV-Rply at 80E500D4 sys-svc (none) msgbk 062FF388
```

VDEVBK Subcommand



Purpose

The VDEVBK subcommand locates, analyzes, and displays a virtual device block.

Operands

userid
is the one- to eight-character owner of the virtual device. Note that the user ID SYSTEM is not allowed.

devnum
is the one- to four-digit hexadecimal virtual device number. The VDEVBK is located by a scan of the virtual device radix tree.

AT *storaddr*
specifies the one- to eight-significant digit hexadecimal logical VDEVBK address directly. It must be on a doubleword boundary.

Usage Notes

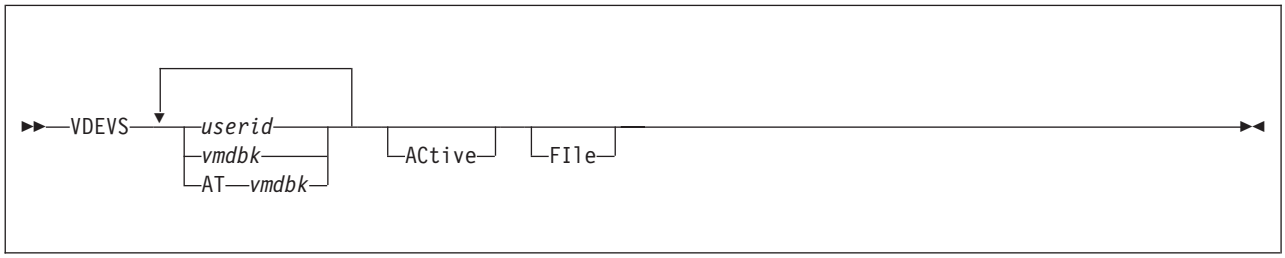
- This subcommand is supported only for CP dumps.

Examples

Typical output from the VDEVBK subcommand:

```
>>> vdev operator 0190
Device number 0190 Subchannel number 0009
VDEVBK at 01C3F310

>>> vdev at 01C3F310
Device number 0190 Subchannel number 0009
VDEVBK at 01C3F310

>>> dts bits on
complete

>>> vdev operator 0190
  Bits  defined in RDEVCLAS         (04)
    04 CLASDASD DIRECT ACCESS STORAGE DEVICE CLASS
  Bits  defined in RDEVDFLG         (82)
    80 RDEVAUTO 370X - AUTO LOAD/DUMP ACTIVE
    02 RDEVMDCP DASD - Caching in MDC enabled for devices with MDC on.
       (DFLTOFF) See RDEVHSID for explanation.
    0 RDEVPSUP TERM - PRINT SUPPRESS AVAILABLE
Device number 0190 Subchannel number 0009
  Bits  defined in VDEVSTAT         (00)
VDEVBK at 01C3F310
```

VDEVS Macro



Purpose

The VDEVS macro displays information about the virtual devices belonging to a user.

Operands

userid
: is the one- to eight-character user ID of the user whose virtual devices you want to display.

vmdbk
AT *vmdbk*
: is the one- to eight-significant digit hexadecimal logical address of the VMDBK you want to display.

ACtive
: specifies that you only want to see VDEVs whose corresponding real device has active I/O (RDEVAIOR is not 0).

FIle
: causes the output to be placed in a file called "dumpname VDEVS A1". If the file already exists, the new information will be appended to the existing file.

Usage Notes

1. VDEVS issues the VDEVBK subcommand for each virtual device to be displayed. If there is a corresponding real device, it issues the RDEVBK subcommand as well.
2. You can specify any one parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
3. If a user ID can be considered an address, use the VMDBK address.
4. This macro is supported only for CP dumps.

See also the "RDEVBK Subcommand" on page 118 and the "VDEVBK Subcommand" on page 158 for related information.

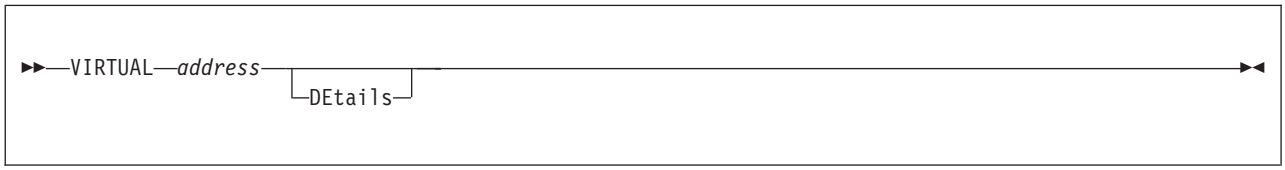
Examples

Typical output from the VDEVS macro:

```
>>> vdevs autolog1
---> processing devices for user AUTOLOG1, vmdbk@ 01D15000

  Control blocks for device number 0009
```

```
    Device number 0009 Subchannel number 0000
    VDEVBK at 01D18190

  Control blocks for device number 000C
    Device number 000C Subchannel number 0001
    VDEVBK at 01D19EC8

  Control blocks for device number 000D
    Device number 000D Subchannel number 0002
    VDEVBK at 01D19C80

  Control blocks for device number 000E
    Device number 000E Subchannel number 0003
    VDEVBK at 01D19A38

  Control blocks for device number 0190
    Device number 0190 Subchannel number 0004
    VDEVBK at 01D196D0

    Device number B702 Subchannel number 08BA
    RDEVBK          at 00C9DBA8

  Control blocks for device number 0191
    Device number 0191 Subchannel number 0005
    VDEVBK at 01D19590

    Device number B702 Subchannel number 08BA
    RDEVBK          at 00C9DBA8

6 devices for user AUTOLOG1 at 01D15000
```

VIRTUAL Macro



Purpose

The VIRTUAL macro displays a user ID and a virtual address associated with a given real address.

Operands

address
is a one- to sixteen-significant digit hexadecimal real address.

DEtails
indicates that intermediate information about the translation should be displayed.

Usage Notes

1. You may specify *address* as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. This macro is supported only for CP dumps.

Examples

Typical output from the VIRTUAL macro:

```
>>> virtual 94240F3
Page owned by OPERATOR
Virtual page = 00000000_00001000

>>> virtual 94240F3 details
   64-bit mode
   input address is           00000000094240F3
   frmte is at                0000000080128480
   frmte:                     00000000017BBB10 00000000801D57C0
                              000000000ECDA808 01ECE00080240000
   type in frmcsb0 is         80
   code in frmcsb1 is         24
   frmcsb1 in bits is         00100100
   PTE is                     000000000ECDA808
   PGMBK is at                000000000ECDA000
   user page
   vmdbk from 0000 is         017BB000
   virt addr from 0008 is     0000000000000000
   status from 0020 is        00000000 shared=0
   displ into page table      0000000000000008
   page table entry number    0000000000000001
   page number in segment     01
   virt addr from 0008 is     0000000000000000
   final virtual address is   00000000_00001000
Page owned by OPERATOR
Virtual page = 00000000_00001000
```

VMDBK Subcommand

PI

Notes:

1 The default is to include all VMDBKs in the global cyclic list.

Purpose

The VMDBK subcommand is used to determine the VMDBK address for a given user ID, the user ID for a given VMDBK address, or to display all the VMDBKs in the global cyclic list.

Operands

userid
is the one- to eight-character user ID that displays and analyzes selected fields in the base VMDBK of the specified user ID.

AT *address*
is the one-to eight-significant digit hexadecimal logical address that displays the user ID for the VMDBK at *address*.

Usage Notes

1. You may specify any single parameter as an underscore character ("_"). Place the cursor on an address in the file area of the screen, and press ENTER. The underscore in the subcommand will be replaced by the value pointed to by the cursor.
2. See also the "VMDBKS Macro" on page 163 and the "VMDSCAN Macro" on page 164 for related information.
3. To halt a long-running VMDBK command, press the PA1 key to break out of full screen mode and then enter the HI command.
4. This subcommand is supported only for CP dumps.

Examples

Typical output from the VMDBK subcommand:

```
>>> vmdbk operator
OPERATOR VMDBK at 770B4000

>>> vmdbk at 770B4000
User is OPERATOR
```

PI end

VMDBKS Macro

PI

Purpose

The VMDBKS macro produces output similar to the VMDBK subcommand with no parameters, except that secondary VMDBKs for all users are also included.

Usage Notes

- This macro is supported only for CP dumps.

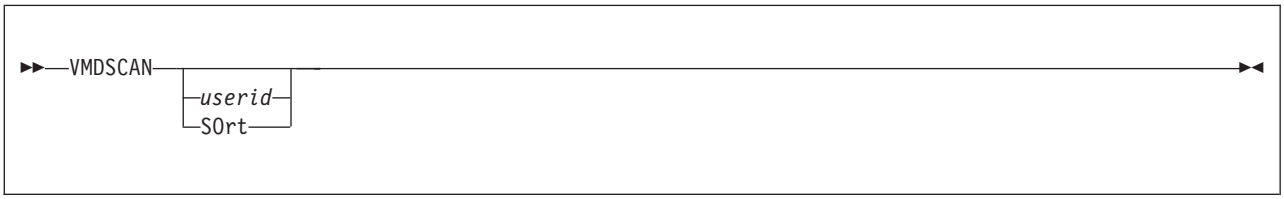
Examples

Typical output from the VMDBKS macro:

```
>>> vmdbks
SYSTEM   base VMDBK at 04500000
SYSTEM   2ary VMDBK at 04501000
OPERATOR base VMDBK at 770B4000
47249A   base VMDBK at 719C6000
CSERVOX1 base VMDBK at 44955000
CFTRECV1 base VMDBK at 041DE000
AXTEST   base VMDBK at 3D450000
AXTEST   2ary VMDBK at 34456000
AXTEST   2ary VMDBK at 34455000
CFTH302  base VMDBK at 34898000
LOGV0013 base VMDBK at 344F5000
CFTHU00  base VMDBK at 7436E000
```

PI end

VMDSCAN Macro



Purpose

The VMDSCAN macro displays information from the dump on all users or a single user.

This macro gives more information than the VMDBKS macro or VMDBK subcommand issued with no parameters.

Operands

userid
is the one- to eight-character user ID for which data is displayed. A VMDBK address is not allowed in this field.

SOrt
indicates the list will be sorted by user ID.

Usage Notes

1. If the user ID you want to enter is SO, SOR, or SORT, use the SORT keyword first; then specify the appropriate *userid*.
2. Information about all of the VMDBKs for the given *userid* is displayed.

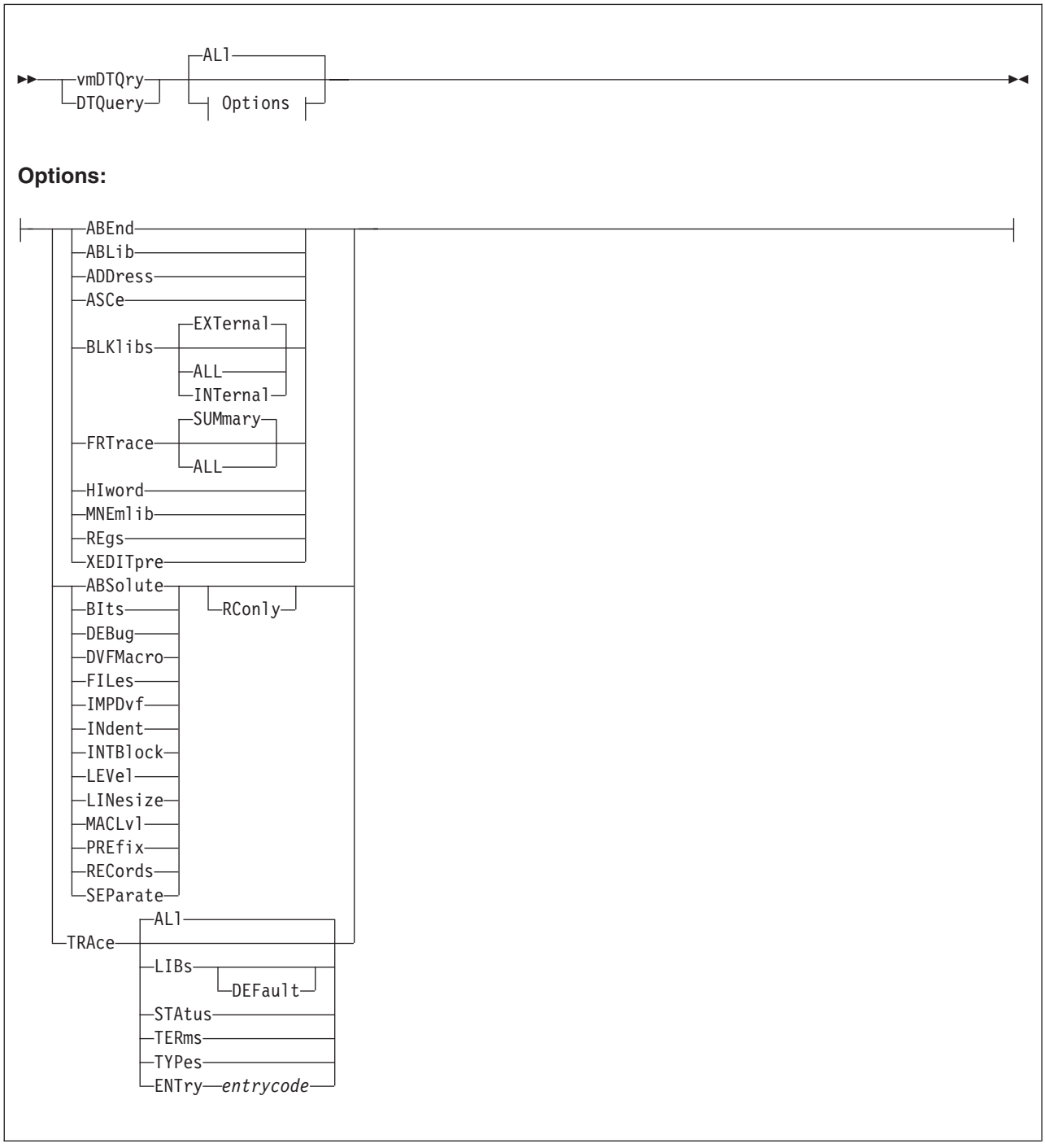
   See also the "VMDBK Subcommand" on page 162 and the "VMDBKS Macro" on page 163 for related information.
3. This macro is supported only for CP dumps.

Examples

Typical output from the VMDSCAN macro is as follows:

```
>>> vmdscan
SYSTEM   VMDBK at 04500000   System      Dispatch list   Suspended
         VMDBK at 04501000   Prototype   not in list     not in list
OPERATOR VMDBK at 770B4000   User Base   Dormant list    Idle
47249A   VMDBK at 719C6000   User Base   Dormant list    Idle
CSERVOX1 VMDBK at 44955000   User Base   Dormant list    Idle
CFTRECV1 VMDBK at 041DE000   User Base   Dormant list    Idle
AXTEST   VMDBK at 3D450000   User Base   Dispatch list   Suspended
         VMDBK at 34456000   Prototype   not in list     not in list
         VMDBK at 34455000   Prototype   not in list     not in list
CFTH302  VMDBK at 34898000   User Base   Dormant list    Idle
```

VMDTQRY Subcommand



Purpose

The VMDTQRY and DTQUERY subcommands are the preferred way to get information about the VM Dump Tool, the operating environment, and the actual dump file. All operands of QUERY that have been supported in the past will continue to be supported without change. New operands will be added only to VMDTQRY and DTQUERY, which are preferred over QUERY.

Operands

ABEnd
indicates that the abend code from the dump should be displayed (or VMDUMP for VM Dump, or SADUMP or SAD001 for Stand-alone Dump). This is a subset of the information available from the CODE Subcommand, but with a simpler output format to make it easier for a macro to deal with. The meaning of the abend code can be found with the DESCRIBE macro.

ABLib
displays the filename of the abend library used by the DESCRIBE subcommand.

ADDress
indicates that the default addressing mode for VM Dump Tool subcommands and macros should be displayed. The response will be one of LOGICAL, REAL, or USER. See also the VMDTSET ASCE subcommand.

ALl
specifies that all VMDTSET values are to be displayed.

ASCe
indicates that the current Address Space Control Element used for translating User-defined addresses be displayed.

BLKlibs
displays the search sequence of block libraries that are used by the BLOCK and ANALYZE subcommands.

EXTernal
displays the list of VMDTDATA files in the order they will be searched. This is the default if no option is specified.

INTernal
displays the description(s) of any block libraries internal to the dump in the order they will be searched.

ALL
displays INTernal responses followed by EXTernal responses.

FRTrace
indicates that information about the Function Related trace tables found in the dump should be displayed.

SUMmary
indicates the output should include only a count of the Function Related trace tables in the dump. This is the default.

ALL
indicates that information about all Function Related trace tables found in the dump should be displayed. This includes tokens and names of all Function Related trace tables.

HIword
Returns the current HIword setting. This is used to determine how to display double-word addresses when the high order full-word of the address is zero.

Responses include:

FULL
the display will consist of the full double-word address with an underscore (_) between the two words.

BLANK
the display will consist of only the underscore (_) and the second full-word.

SINGLE0
the display will consist of a single zero, the underscore (_), and the second full-word.

MNEmlib
displays the filename of the mnemonic library used by the INSTR subcommand.

REgs
Indicates how register values are to be displayed from the GREG subcommand.

Responses include:

MAP
the value of each register is mapped to a module and displacement.

NOMAP
Register values are not to be mapped to modules.

SHORT
Register values should be assumed to be 4 bytes long.

LONG
Register values should be assumed to be 8 bytes long.

XEDITpre
displays the current setting for the XEDIT prefix string.

ABSolute
displays the current setting. ABSOLUTE ON indicates that references to location 0-FFF (0-1FFF in a 64-bit dump) on the failing processor are not prefixed by the contents of the prefix register.

BIts
Indicates whether bit definitions are to be displayed automatically with VM Dump Tool subcommands including FRAME and a number of trace types.

Responses include:

ON bit and code meanings will be displayed.

OFF
bit and code meanings will not be displayed

DEBug
displays or returns the current DEBUG setting.

DVFMacro
specifies that the value of the SET DVFMACRO input is to be displayed and returned.

FILes
indicates the total number of CMS files in the dump.

IMPDvf
specifies that the value of the SET IMPDV input is to be displayed and returned.

INDENT
displays or returns the current indent setting.

INTBlock
indicates whether or not block libraries internal to the dump, if any, will be used by ANALYZE and BLOCK when resolving a symbol.

LEVel
specifies that the release level of the VM Dump Tool, itself, should be displayed.

LINesize
specifies that the length of the current output line should be displayed.

MACLvl
specifies that the macro nesting level is to be displayed and returned. The response from issuing the VMDTQRY subcommand is 0. When issued from a macro, it is 1 or greater to indicate the nesting level.

PREfix
displays the current. PREFIX ON indicates that references to location 0-FFF (0-1FFF in a 64-bit dump) for the failing processor are prefixed by the contents of the prefix register.

RECords
displays the total number of records in the dump.

SEParate
displays the status of the SEPARATE option. If ON, a blank line will be added after each command issued from the command line.

PI **RConly**
returns the result as a return code. RC=0 indicates OFF; RC=1 indicates ON. **PI** **end**

TRACE Operands

TRAce
specifies that only the TRACE-related information should be displayed.

ALl
specifies that all trace-related information should be displayed.

LIBs
specifies that only information about the trace definition libraries should be displayed.

DEFault
displays the filename of the default trace formatting library used by the TRACE subcommand.

STAtus
specifies that only the first line of trace-related information should be displayed.

TERms
specifies that only the list of trace terms should be displayed.

TYPes
specifies that only the list of trace types should be displayed.

ENTry *entrycode*
specifies that only the formatting information for the trace entry code specified be displayed.

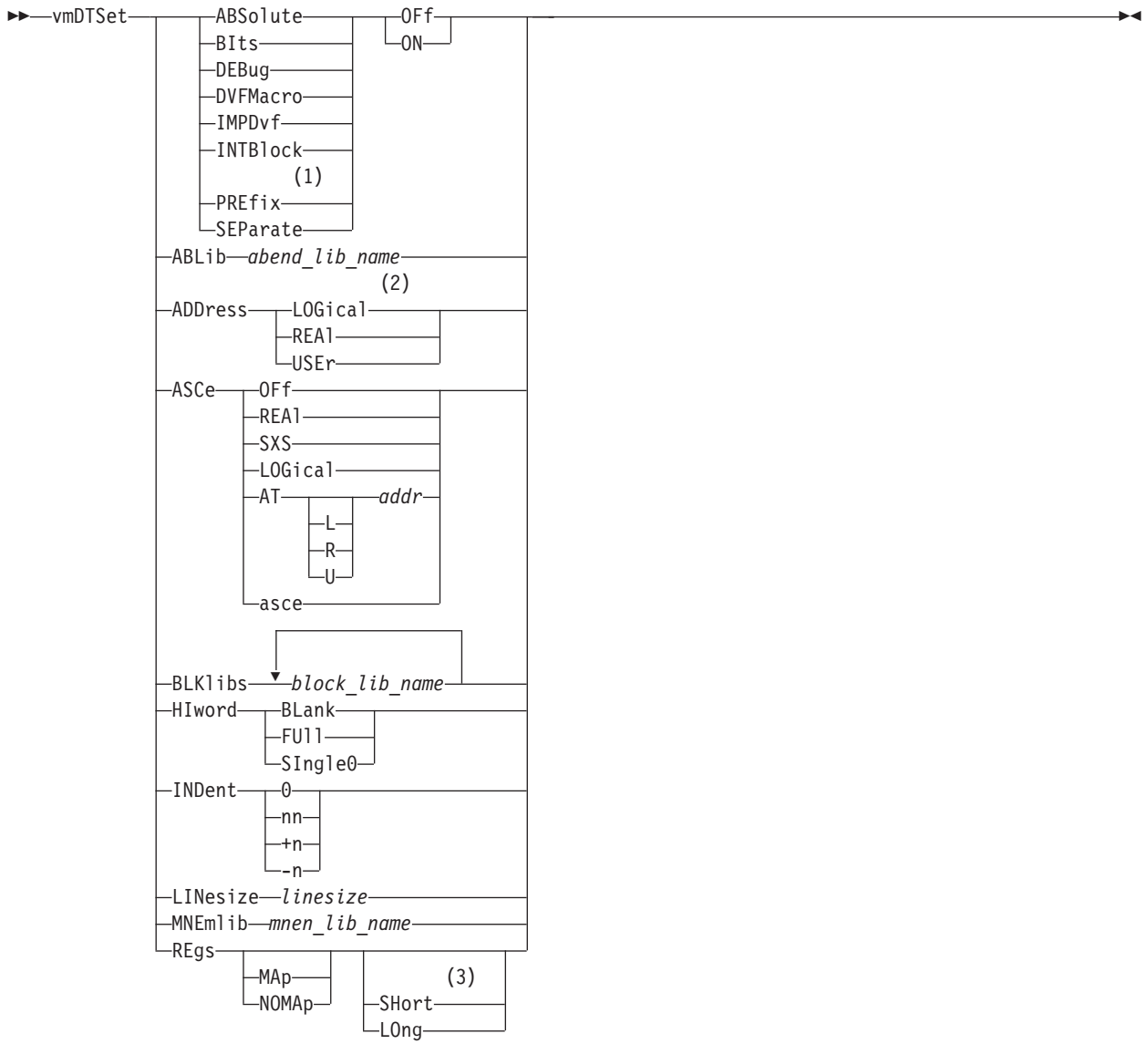
Examples

Typical output from the VMDTQRY subcommand:

```
>>> vmdtqry abend
ABEND    set to SVC002

>>> vmdtqry address
ADDRESS  set to LOGICAL

>>> dtq asce
ASCE     set to 00000000 00000020

>>> dtq trace
TRACE    set to ADDRESS CPU/TIME NOCODE DATA NOSLASH ASSERT
    LIB  set to Default> HCQTR540
    TERM set to ADDRESS  ASCBK    ASTE     CARBK    CCTBK    COMBK
                CPEBK    DUMPCODE ESW      EXTCODE  FCTBK    FRMTE
                HASHID   IACCODE  IORBK    IPARML   IUCVB    IXBLK
                LATBK    LDEVNO   LNKBK    LOCBK    LOCK     LOCTHRED
                LSCH     MDEBK    MODULE   MSGBK    PATH     PGMBK
                PGMCODE  PTHBK    RDEV     RDEVNO   RSCH     SID
                SNABK    SUBCH    SVCCODE  SYSSVCCD TCHBK    TCHKEY
                TCHTKFMT TSKBK    VDEV     VDEVNO   VMDBK    VSCH
                WEIBK    XCRBK    XITBK
    TYPE set to APPC     CALL     CCALL    CCS      CONTROL  EXIT
                EXT      EXTERNAL FREE     I/O      IO       IUCV
                LI/O     LIO      MDC      PC       RI/O     RIO
                SCSI     SCSILOCK SENSE    SIE      SIGP     SPINLOCK
                STACK    STORAGE  VCTC     VI/O     VIO      VSTORAGE

>>> dtq trace status
TRACE    set to NEW ADDRESS CPU/TIME NOCODE DATA NOSLASH

>>> dtq trace lib
     LIB set to Default> HCQTR540

>>> dtq  frt all
     FRTRACE  set to  3
     Token    Status   Pages Current     Caller      Name
     00AFA041 Open     0003 00AF90C0 HCPDST+22C      Table.A
     00AFA081 Open     0003 00AF60C0 HCPDST+22C      MyTable
     00AFA0C1 Open     0003 00AF20C0 HCPDST+22C      John

>>> dtq frt summary
     FRTRACE set to 3

>>> dtq trace terms
    TERM set to ADDRESS  ASCBK    ASTE     CARBK    CCTBK    COMBK
                CPEBK    DUMPCODE ESW      EXTCODE  FCTBK    FRMTE
                HASHID   IACCODE  IORBK    IPARML   IUCVB    IXBK
                IXBLK    LATBK    LDEVNO   LNKBK    LOCBK    LOCTHRED
                LSCH     MDEBK    MODULE   MSGBK    PATH     PGMCODE
                PTHBK    RDEV     RDEVNO   RSCH     SID      SNABK
                SUBCH    SVCCODE  SYSSVCCD TCHBK    TCHKEY   TCHTKFMT
                TSKBK    VDEV     VDEVNO   VMDBK    VSCH     WEIBK
                XCRBK    XITBK

>>> dtq trace types
    TYPE set to APPC     CALL     CCALL    CCS      CONTROL  EXIT
                FREE     I/O      IO       IUCV     LI/O     LIO
                LOCK     MDC      PC       RI/O     RIO      SCSI
                SENSE    SIE      STACK    STORAGE  VCTC     VI/O
                VIO      VSTORAGE

>>> dtq trace entry 0600
from library HCQTRACE
```

```
  'Obtain' DEC 10 04
  'dw' PCHAR 0D 03
  'at' ADDRESS 14 04
  'by' MODULE 1C 04
  'vmdbk' VMDBK 18 04
  USERID 18 04

>>> dtq ablib
ABLIB    set to HCQAB540

>>> dtq blklibs
BLKLIBS  set to HCQD8540 HCQD8530 HCQD8520 HCQB8510

>>> dtq mnemlib
MNEMLIB  set to HCQMN540

>>> dtq separate
SEPARATE set to ON

>>> dtq trace lib
    LIB  set to Default> HCQTR540

>>> dtq
ABEND    set to SVC002
ABLIB    set to HCQAB540
ABSOLUTE set to OFF
ADDRESS  set to LOGICAL
ASCE     set to 00000000 00000020
BITS     set to OFF
BLKLIBS  set to HCQD8540 HCQD8530 HCQD8520 HCQB8510
DEBUG    set to OFF
DVFMACRO set to OFF
FILES    set to 156
FRTRACE  set to 3
HIWORD   set to FULL
IMPDVF   set to OFF
INDENT   set to   0
INTBLOCK set to ON
LEVEL    set to 540
LINESIZE set to 72
MACLVL   set to   0
MNEMLIB  set to HCQMN540
PREFIX   set to ON
RECORDS  set to 1039473
REGS     set to MAP      LONG
TRACE    set to ADDRESS CPU/TIME NOCODE DATA NOSLASH ASSERT
    LIB  set to Default> HCQTR540
    TERM set to ADDRESS  ASCBK    ASTE     CARBK    CCTBK    COMBK
                CPEBK    DUMPCODE ESW      EXTCODE  FCTBK    FRMTE
                HASHID   IACCODE  IORBK    IPARML   IUCVB    IXBLK
                LATBK    LDEVNO   LNKBK    LOCBK    LOCK     LOCTHRED
                LSCH     MDEBK    MODULE   MSGBK    PATH     PGMBK
                PGMCODE  PTHBK    RDEV     RDEVNO   RSCH     SID
                SNABK    SUBCH    SVCCODE  SYSSVCCD TCHBK    TCHKEY
                TCHTKFMT TSKBK    VDEV     VDEVNO   VMDBK    VSCH
                WEIBK    XCRBK    XITBK
    TYPE set to APPC     CALL     CCALL    CCS      CONTROL  EXIT
                EXT      EXTERNAL FREE     I/O      IO       IUCV
                LI/O     LIO      MDC      PC       RI/O     RIO
                SCSI     SCSILOCK SENSE    SIE      SIGP     SPINLOCK
                STACK    STORAGE  VCTC     VI/O     VIO      VSTORAGE
XEDITPRE set to XEDIT
```

VMDTSET Subcommand

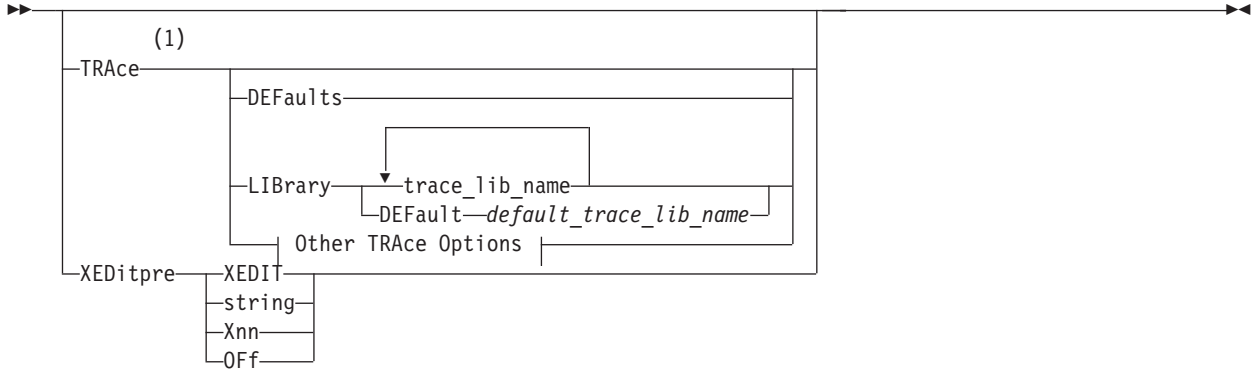


Notes:

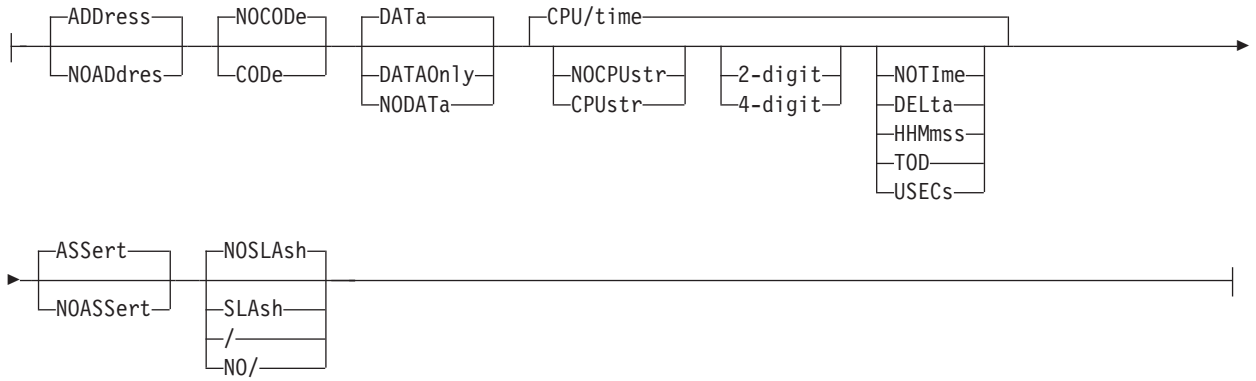
1 The default is ON for the failing CPU, OFF for all others.

2 The default is LOGical for CP dumps after version 5, release 1.0 when the dump is of CP and was running in zArchitecture mode. For previous releases and other types of dumps, the default is REAl.

3 The default depends upon the size of registers in the dump.



Other TRAce Options:



Notes:

1 Following vmDTSet TRAce, you must choose at least one operand from the options shown. You can choose up to one operand in each set.

Purpose

The VMDTSET subcommand is available to let the user tailor the output and operation of some subcommands and macros.

Operands

ABSolute
specifies if locations 0-FFF (0-1FFF in a 64-bit dump) on the failing processor should be referenced with absolute addresses. Note that ABSOLUTE and PREFIX are opposite and affect the same setting.

ON indicates that references to low storage will be with absolute address (no prefixing).

OFF
indicates that prefixing should be applied to references to low storage of the failing processor.

DEBug
specifies whether error messages and non-zero return codes produced from running a VM Dump Tool macro should be displayed on the virtual machine console.

DVFMacro
specifies whether ADDRESS SCAN input is accepted or rejected. Options are:

ON specifies that ADDRESS SCAN input will be accepted and processed.

OFf
specifies that ADDRESS SCAN input will be rejected with message HCQ106E.

IMPDvf
specifies whether command line input is used to search for a SCAN macro. Options are:

ON specifies that SCAN macros will be included in the search sequence when an input subcommand is not recognized as a VMDT macro, a VM Dump Tool subcommand, or an XEDIT macro.

OFf
specifies that command line input should not be used to search for a SCAN macro.

INTBlock
specifies whether or not block libraries internal to the dump, if any, will be used by ANALYZE and BLOCK when resolving a symbol.

ON specifies internal block libraries, if they exist, will be used. This is the default.

PI

Format of internal block libraries is documented in Appendix D, "Format of Block Data," on page 219. For details on how to include a block library so the VM Dump Tool can locate it within a dump, see the prologue to CP module HCPHCQ.

PI end

OFf
specifies that internal block libraries, if they exist, will be ignored and all symbols will be resolved via external VMDTDATA files.

SEParate
specifies whether a blank line should be added to the dump session after the output of each VM Dump Tool subcommand which is issued from the command line.

ON specifies that a blank line should be added after each command after each VM Dump Tool subcommand issued from the command line.

OFf
specifies that blank lines should not be added.

ABlib
specifies that the filename of the library to be used by the DESCRIBE subcommand follows.

abend_lib_name
specifies the filename of the library to be used by the DESCRIBE subcommand. Only one such name can be supplied. The filetype is always VMDTDATA.

ADDress
allows you to specify the default addressing mode for subcommands which can handle more than one. (See "Address Spaces and the VM Dump Tool" on page 10.)

LOGical
indicates that the default address space should be the logical address space.

REAl
indicates that the default address space should be the real address space.

USEr
indicates that the default address space should be the user-defined address space. See the VMDTSET ASCE subcommand for further information about this operand.

ASCe
allows you to specify the address space to be used for references to the 'user-defined' address space through SET ADDRESS USER or the U addressing prefix character.

OFf
REAl
indicate that the address should not be translated. That is, it is a real address.

LOGical
SXS
indicates that the address should be interpreted as being in the System eXecution Space (which is the same as the Logical Address Space).

L, R, or U
is an optional address space prefix character. Use L for Logical, R for Real, or U for User-defined. There should be no space between the prefix character and *addr*. If not specified, the address space specified or defaulted by the VMDTSET ADDRESS subcommand will be used.

AT *addr*
specifies the one- to sixteen-significant digit hexadecimal address where the subject eight-byte ASCE should be loaded from. The L, R, or U indicates that this addressing mode for this reference defaults to that set by VMDTSET ADDRESS. The default can be overridden or the address mode can be specified directly by including an L, R or U prefix character on the address.

asce
specifies the one- to sixteen-significant digit hexadecimal value which should be used to control translation of addresses defined to be in the User Address Space. If you specify fewer than sixteen digits, it will be padded on the left with zeros.

BIts
specifies whether bit definitions are to be displayed automatically with VM Dump Tool subcommands, macros and a number of trace formatters. Options are:

ON bit and code meanings are displayed.

OFf
bit and code meanings are not displayed

BLKlibs
specifies that one or more filenames of libraries to be used by the BLOCK and ANALYZE subcommands follows.

block_lib_name
specifies the filename of one library to be used by the BLOCK and ANALYZE subcommands. Up to 12 such filenames may be supplied. The filetype is always VMDTDATA.

HIword
specifies how to display double-word register values when the high order full-word of the 64-bit data is zero. Options are:

BLank
the display will consist of only the underscore (_) and the second full-word.

FUll
the display will consist of the full double-word data with an underscore (_) between the 2 words.

SIngle0
the display will consist of a single zero, the underscore (_), and the second full-word.

INDent
Sets the offset from the left margin for output. It is usually used only within macros. Options are:

nn is a one- to two-digit decimal number that specifies an absolute indentation of *nn* spaces.

+*nn*
specifies an increased indentation of *nn* spaces.

-*nn*
specifies a decreased indentation of *nn* spaces.

LINesize *linesize*
specifies maximum number of characters to be placed on the output line for display. The maximum value allowed is 255.

MNEmlib
specifies that the filename of the library to be used by the INSTR subcommand follows.

mnem_lib_name
specifies the filename of the library to be used by the INSTR subcommand. Only one such name can be supplied. The filetype is always VMDTDATA.

PREfix
specifies if locations 0-FFF (0-1FFF in a 64-bit dump) on the failing processor should be prefixed per the prefix register for that processor. Note that PREFIX and ABSOLUTE are opposite and affect the same setting.

ON indicates that prefixing should be applied to references to low storage.

OFF
indicates that prefixing should not be applied to references to low storage of the failing processor.

REgs
specifies how register values are to be displayed by the GREGS subcommand. Options are:

MAp
the value of each register is mapped to a module and displacement.

NOMAp
register values are not to be mapped to modules.

SHort
register values are assumed to be four bytes long.

LOng
register values are assumed to be eight bytes long.

XEDitpre
defines an XEDIT prefix string, which allows you to pass to XEDIT a command that would normally be interpreted directly by the VM Dump Tool. Options are:

string
is a one- to eight-character EBCDIC string that the VM Dump Tool recognizes as the XEDIT prefix string. This value cannot be any of the following: FILE, FFile, SAVE, SSave, Quit, QQuit (or abbreviations thereof), OFF, or Xnn where nn is a valid hexadecimal value. The default is the string XEDIT.

X*nn*
is the hexadecimal value of a one-character string that VM Dump Tool recognizes as the XEDIT prefix string.

OFf
specifies that there is no XEDIT prefix string.

TRACE Operands

ADDress
specifies that the address of the trace entry should be displayed.

NOADdres
specifies that the address of the trace entry should not be displayed.

CODe
specifies that the trace entry code for each trace entry should be displayed.

NOCODe
specifies that the trace entry code for each trace entry should not be displayed.

DEFaults
specifies that all TRACE settings should be reset to their original values.

DATAOnly
specifies that the header portion of each line should be suppressed and that only the data portion of each trace entry should be displayed.

CPU/time
specifies that the CPU number should be displayed for each trace entry if

MERGE is also specified, or the time to the nearest second should be displayed for each trace entry if MERGE is not also specified. The minimum abbreviation is CPU/.

CPUstr
specifies that the CPU string should be displayed ahead of a CPU number for each trace entry.

NOCPUstr
specifies that the CPU string not be displayed ahead of a CPU number for each trace entry.

2-digit
specifies that the CPU number should be displayed as a two digit number.

4-digit
specifies that the CPU number should be displayed as a four digit number.

NOTIme
specifies that no time value should be displayed with each trace entry.

DELta
specifies that the difference in microseconds from the previously displayed trace entry should be displayed with each trace entry.

HHMmss
specifies that the time to the nearest second should be displayed for each trace entry.

TOD
specifies that the 16-digit hexadecimal Time-Of-Day value should be displayed with each trace entry.

USECs
specifies that the time to the nearest microsecond should be displayed for each trace entry.

DATa
specifies that the data portion of each trace entry should be displayed.

NODATa
specifies that the data portion of each trace entry should not be displayed. Only the header information will be displayed.

SLAsh
/
specifies that the larger type-75 trace entries be preceded with a slash character.

NOSLAsh
NO/
specifies that the larger type-75 trace entries should not be specially marked.

ASSert
specifies that trace entries generated by the CP HCPASERT facility should be included in the formatted trace table.

NOASSert
specifies that trace entries generated by the CP HCPASERT facility should be skipped over when formatting the trace table.

LIBrary
specifies that a trace formatting library name or operand follows.

trace_lib_name
specifies the filename of one library to be used by the TRACE subcommand. Up to 12 such filenames may be supplied. The filetype is always VMDTDATA.

DEFault
specifies that the name of the default trace formatting library follows.

default_trace_lib_name
specifies the filename of the default trace formatting library to be used by the TRACE subcommand. Only one such name can be supplied. The filetype is always VMDTDATA.

Usage Notes

1. Some subcommands and macros have operands which can temporarily override the BITS or NOBITS setting for the duration of that subcommand or macro.
2. Only TRACE and VMDTQRY honor this length at this time.
3. If the setting of LINESIZE is too small (and that size varies by trace entry code), then TRACE will display the minimum amount that it can on each line, but the total length of the line may be longer than the linesize that was set.
4. Settings after using VMDTSET TRACE DEFAULTS:

```
>>> dts trace defaults
complete
>>> dtq trace status
TRACE    set to NEW ADDRESS CPU/TIME NOCODE DATA NOSLASH
```

5. The CPU/TIME option is a special combination of CPU options and time options, which otherwise are two separate options. When changing either the CPU format or the time format from the CPU/TIME setting, the other (time for the CPU option, CPU for the time option) must be set also or it will not be displayed.

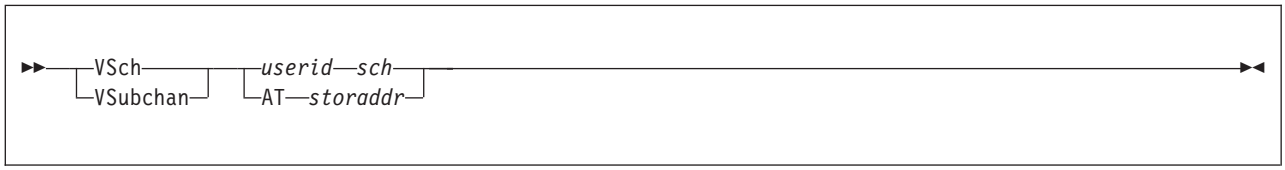
Examples

Typical output from the VMDTSET subcommand:

```
>>> vmdtset address logical
complete

>>> vmdtset prefix on
complete

>>> dts xeditpre xedit
complete

>>> dts asce real
ASCE     set to 00000000 00000020

>>> dts trace addr notime 2-digit
complete

>>> dtq
ABEND    set to SVC002
ABLIB    set to HCQAB540
ABSOLUTE set to OFF
ADDRESS  set to LOGICAL
ASCE     set to 00000000 00000020
BITS     set to OFF
BLKLIBS  set to HCQD8540 HCQD8530 HCQD8520 HCQB8510
DEBUG    set to OFF
DVFMACRO set to OFF
```

```
FILES    set to 156
HIWORD   set to FULL
IMPDVF   set to OFF
INDENT   set to   0
INTBLOCK set to ON
LEVEL    set to 540
LINESIZE set to 72
MACLVL   set to   0
MNEMLIB  set to HCQMN540
PREFIX   set to ON
RECORDS  set to 1039473
REGS     set to MAP      LONG
SEPARATE set to ON
TRACE    set to ADDRESS CPU/TIME NOCODE DATA NOSLASH ASSERT
    LIB  set to Default> HCQTR540
    TERM set to ADDRESS  ASCBK    ASTE     CARBK    CCTBK    COMBK
                CPEBK    DUMPCODE ESW      EXTCODE  FCTBK    FRMTE
                HASHID   IACCODE  IORBK    IPARML   IUCVB    IXBLK
                LATBK    LDEVNO   LNKBK    LOCBK    LOCK     LOCTHRED
                LSCH     MDEBK    MODULE   MSGBK    PATH     PGMBK
                PGMCODE  PTHBK    RDEV     RDEVNO   RSCH     SID
                SNABK    SUBCH    SVCCODE  SYSSVCCD TCHBK    TCHKEY
                TCHTKFMT TSKBK    VDEV     VDEVNO   VMDBK    VSCH
                WEIBK    XCRBK    XITBK
    TYPE set to APPC     CALL     CCALL    CCS      CONTROL  EXIT
                EXT      EXTERNAL FREE     I/O      IO       IUCV
                LI/O     LIO      MDC      PC       RI/O     RIO
                SCSI     SCSILOCK SENSE    SIE      SIGP     SPINLOCK
                STACK    STORAGE  VCTC     VI/O     VIO      VSTORAGE
XEDITPRE set to XEDIT
>>> dts lin 80
complete

>>> dtq lin
LINESIZE set to 80

>>> dts lin 72
complete

>>> dtq lin
LINESIZE set to 72

(When LINESIZE is set too small for TRACE):

>>> dts lin 10
complete

>>> trace for 3
7FB60460 10:54:40
                  Enter-wait-state
                  wait-mask 6F000000_00000000
7FB60440 10:54:40
                  Call
                  fr HCPHAS+1084
                  to HCPTRODL
                  cpebk 0D63AA00
                  iac AR-Mode
                  parm 0F546E58
7FB60420 10:54:40
                  Rtrn
                  to HCPHAS+1084
                  fr HCPTRO+CE
                  cpebk 43B2DC00
                  iac Primary
                  cc1
                  rc>FFF
```

VSCH Subcommand



Purpose

The VSCH subcommand locates, analyzes, and displays a virtual device block.

Operands

userid
is the one- to eight-character user ID for which VDEVBK data is displayed.

sch
specifies the one- to four-digit hexadecimal virtual subchannel number. The VDEVBK is located by a scan of the virtual subchannel radix tree.

AT *storaddr*
specifies the one- to eight-significant digit hexadecimal logical address of the block to be displayed.

Usage Notes

This subcommand is supported only for CP dumps.

Examples

Typical output from the VSCH subcommand:

```
>>> vsch operator 0009
Device number 0190 Subchannel number 0009
VDEVBK at 01C3F310

>>> vsch at 01C3F310
Device number 0190 Subchannel number 0009
VDEVBK at 01C3F310

>>> dts bits on
complete

>>> vsch operator 0009
  Bits  defined in RDEVCLAS         (04)
    04 DIRECT ACCESS STORAGE DEVICE CLASS
  Bits  defined in RDEVDFLG         (82)
    80 370X - AUTO LOAD/DUMP ACTIVE
    02 DASD - Caching in MDC enabled for devices with MDC on. (DFLTOFF)
       See RDEVHSID for explanation.
    80 TERM - PRINT SUPPRESS AVAILABLE
Device number 0190 Subchannel number 0009
  Bits  defined in VDEVSTAT         (00)
VDEVBK at 01C3F310
```

Chapter 4. Non-CP Dumps

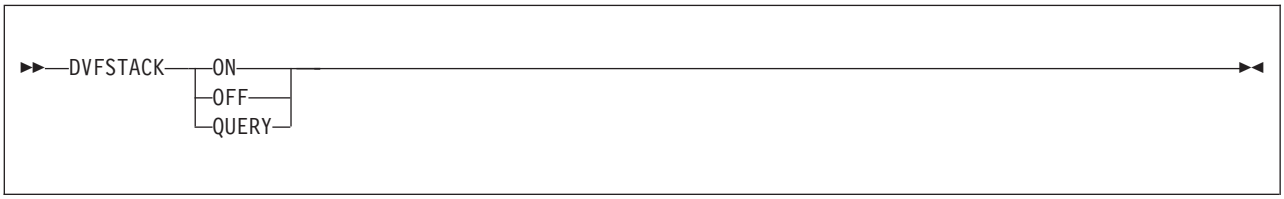
This chapter describes the support provided for non-CP dump handling. This support includes:

- Recognition of a non-CP dump.
- Filtering CP-only VM Dump Tool commands and macros to work only in CP dumps, including changes to VM Dump Tool initialization to not invoke CP-only commands and macros in a non-CP dump.
- DVF primitives, including support for ADDRESS SCAN, the invocation of SCAN macros from VMDT and XEDIT macros, and the invocation of SCAN macros from the VMDT command line. DVF primitives are provided as a migration path to the VM Dump Tool for macros written for DVF users and are not intended to be completely compatible with the DVF equivalents. These subcommands are summarized in Table 3, and described in more detail in the following sections.
- A user exit, the VMDTNCPM VMDT macro, allows you to create a VMDTMAP file for a non-CP dump such that the MAP command and related subcommands function correctly. A sample VMDTNCPM macro is provided. See "Non-CP MAP User Exit" on page 190 for more information.

Table 3. DVF Primitive Subcommands Used Only with a non-CP Dump

Subcommand	Description
DVFSTACK	Directs DUMPSCAN subcommand output to the program stack
FINDStrg	Searches for a particular string of data in the dump from a macro.
INIT	Inserts the name of the dump, dumptype, and date and time the dump was taken into the session file.
NOTE	Inserts text into the dump session.
READStrg	Reads data from the dump, through a direct or indirect address.

DVFSTACK Subcommand



Purpose

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

Operands

ON directs subcommand output to be put on the program stack.

OFF
resets the DVFSTACK on invocation; subcommand output is directed to the dump session file.

QUERY
indicates the current DVFSTACK setting does not change, but returns the setting of DVFSTACK as a return code.

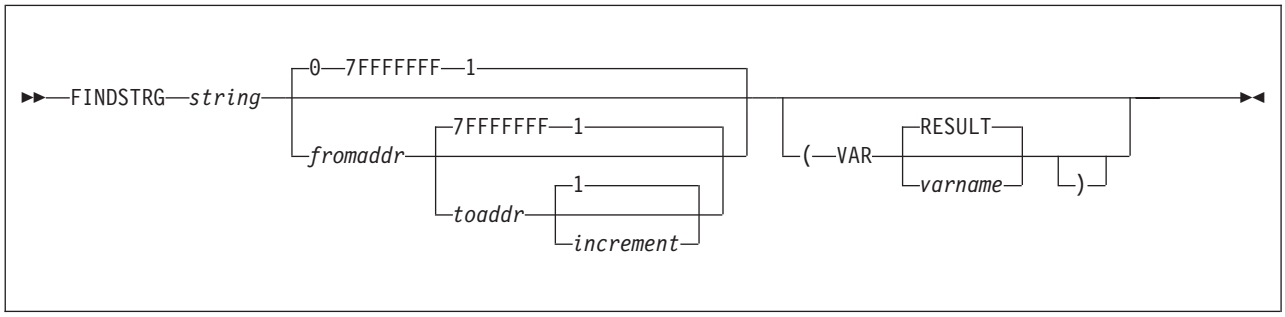
Usage Notes

1. DVFSTACK can be executed only from a macro.
2. This subcommand provides a migration path and is not really compatible with the DVF DVFSTACK subcommand. The normal linkage for the VM Dump Tool output is to put it in the program stack. Whatever remains in the program stack when exiting from a subcommand or macro is added to the dump session.
3. DVFSTACK OFF is accepted as valid, with a return code of 0, but has no effect. After DVFSTACK OFF is issued, a DVFSTACK QUERY will indicate that it is still set to ON.

Return codes

Return Code	Explanation
0	Successful execution
2	You issued a DVFSTACK QUERY and DVFSTACK was set to ON
8	String not found

FINDSTRG Subcommand



Purpose

Use the FINDSTRG subcommand to search for a particular string of data in the dump from a macro. If found, the eight-byte hexadecimal address of the string is returned in a REXX variable.

Operands

string
: is a two- to 128-character (one- to 64-byte) hexadecimal string for which the macro wishes to search. *string* must have an even number of digits.

fromaddr
: is the 31-bit (one- to four-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 (one- to four-byte) hexadecimal ending address for the search. If not specified, this defaults to end at location 7FFFFFFF or the end of the dump. Leading zeros are not required.

increment
: is a one- to four-digit hexadecimal number to change the current address after each match attempt. The valid increment range is from one to 1000 (hexadecimal).

VAR
: indicates that the next parameter 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 variable name.

varname
: is a one- to eight-character user-specified name of a REXX variable where the results of the FINDSTRG subcommand are placed.

Usage Notes

1. This subcommand can be executed only from a macro. An error message is issued if it is entered from the command line.
2. Unlike the LOCATE subcommand or LOCDISP macro, 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 first be converted to hexadecimal.

3. Both the start of the string and end of the string must be within the address range specified by the *fromaddr* and *toaddr* addresses.

   Note: This is different from the DVF function, which requires only the start of the string be within the range of addresses.
4. In order to specify an *increment*, both the *fromaddr* and *toaddr* addresses must be specified.
5. If found, the address of the first byte of the string is placed in a REXX variable, as either RESULT or as 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.
7. The FINDSTRG subcommand does not have a subcommand abbreviation when implemented in the VM Dump Tool. The minimum command abbreviation for the DVF subcommand is FINDSTRG is FINDS.
8. Blanks are not required adjacent to the parentheses.
9. You should consider using the LOCATE, LOCDISP, and SETVAR subcommands to replace existing FINDSTRG macros. The VM Dump Tool FINDSTRG macro is supported only as a migration path from DVF to VM Dump Tool.
10. FINDSTRG invokes LOCATE to do the work. The LOCATE syntax for the *start* operand also applies to the FINDSTRG *from* operand.

Return codes

Return Code	Explanation
0	Successful execution
8	String not found
16	Invalid operands
(other)	number of error message issued

INIT Macro

Purpose

Use the INIT macro to put the name of the dump, the dump type, and the date and time the dump was taken into the session file.

Usage Notes

1. INIT can be executed only from a macro.
2. A macro can use this subcommand to obtain the dump type.
3. Dump types that can be returned are VM for a virtual machine dump, CP for a CP abend or stand-alone dump, or SA for a soft abend dump.
4. Messages 200 and 401 are put into the dump session.

   Note: The other parts of the message identifier are different from DVF in the VM Dump Tool implementation. The messages will have a 3–character component identifier of HCQ for the VM Dump Tool instead of HCS for DVF, and a module id of ACT instead of DSS.
5. You should consider using the DUMPNAME and DUMPTYPE subcommands to replace existing INIT macros. The VM Dump Tool INIT macro is supported only as a migration path from DVF to VM Dump Tool.

Examples

The following is typical output from the INIT command when issued from within a macro: `>>> init`

```
HCQACT200I PROCESSING FILE 2T0L625  DUMP0001 P1 07/16/02 19:39:35
HCQACT401I READY, DUMP TYPE IS CP
```

Return codes

Return Code	Explanation
0	Successful execution

NOTE Subcommand

Notes:

1 DVF accepts other options. They are ignored by the VM Dump Tool.

Purpose

Use the NOTE subcommand to send text output to the dump session.

Operands

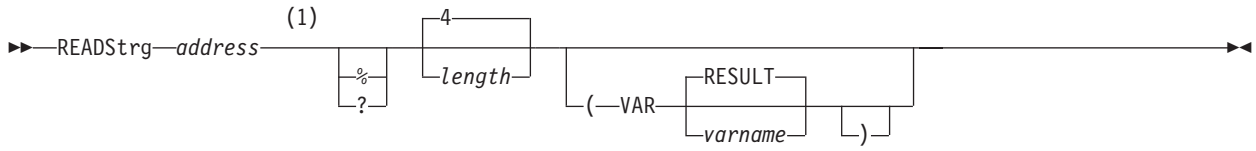
'text'
is the output to be displayed. This includes any blanks between the opening and closing single quotation marks. The maximum length of the text is 255 bytes. If no text is specified, a blank line is displayed. The opening single quotation mark is required. If a closing single quotation mark is not found, the whole text string specified will be displayed.

options
anything following the closing single quotation mark is ignored without an error message.

Usage Notes

1. This subcommand can be issued only from a macro.
2. DVF accepts other options to indicate if the output is to be directed to the terminal or a printer. The VM Dump Tool ignores anything after the closing single quotation mark and directs all output to the user's dump session.

READStrg Subcommand



Notes:

1 A blank is not allowed between *address* and either % or *?*.

Purpose

Use the READSTRG subcommand to read data from the dump from 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 (one– to four–byte) hexadecimal real address from which the data is to be retrieved in the dump. Leading zeros are not required.

% specifies that the address is to be masked to a 24–bit address; the result is used to find the data to be retrieved.

? specifies that the address is to be masked to a 31–bit address; the result is used to find the data to be retrieved.

length
is a one– to four–digit nonzero hexadecimal number indicating the length, in bytes, to be returned to the macro. The valid range is one to 1000. Four is the default length of the data to be returned.

VAR
indicates that the next parameter is the user-specified REXX variable name.

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

varname
is a one– to eight–character user-specified name of a REXX variable where the results of the READSTRG subcommand are placed.

Usage Notes

1. This subcommand can be executed only 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 variable RESULT or the user-specified variable).
3. If only partial data is available in the dump, READSTRG ends prematurely with an error message and return code.
4. Blanks are not required adjacent to the parentheses.

5. You should consider using the SETVAR subcommand to replace existing READSTRG macros. The VM Dump Tool READSTRG macro is supported only as a migration path from DVF to VM Dump Tool.

Return codes

Return Code	Explanation
0	Successful execution
16	Invalid operand
(other)	number of error message issued

Chapter 5. VM Dump Tool User Exits

This chapter describes programs included in the VM Dump Tool that the user can modify.

Default User Profile

The VM Dump Tool profile is run when you invoke the VM Dump Tool. In this file you can change any PF key or XEDIT option set up by the VM Dump Tool.

The preferred name for the VM Dump Tool profile is PROFILE VMDT. For compatibility with previous releases, it will also find and use a file with the name VMDTPROF VMDT or VMDTPROF XEDIT. It searches for the first one in that order, invokes it, and then proceeds with the rest of VM Dump Tool initialization.

For example, to set up a Prefix Area in an XEDIT session, add the following line to your VM Dump Tool profile:

```
'SET PREFIX ON'
```

Note that you should change the defaults at your own risk. Changing some options may cause output to be displayed in unusual ways. The VM Dump Tool assumes that the data is displayed from the top of the screen. If the CURLINE setting is changed, it may be necessary to page forward to see the last few lines of data.

A number of VM Dump Tool subcommands are invoked when a file is entered for the first time. These may be removed or modified as required. Be aware that the program stack is the communication stream for sending output lines back to the dump session. You must save and restore the stack if you generate new stack output that you do not want displayed. Also, be careful not to empty the stack of data that you want displayed.

BLOCK Initialization Profile

The VMDTBPRF XEDIT file is the BLOCK-related user exit.

If the VMDTBPRF XEDIT file is not found during processing, processing continues without error. If the VMDTBPRF XEDIT file is found during processing, certain information is passed to the file, and a string of library file names to search is expected in return.

The parameters passed to VMDTBPRF include the identifier of the dump, the version level of the dump, the mode of CP present in the dump, and the suggested string of library file names to use. The data is passed in the following format:

```
DUMPSPEC <fn ft fm> VER <lvl> MODE <dumpmode> BLKLIBS <list>
```

where:

- *fn* is the file name of the dump
- *ft* is the file type of the dump
- *fm* is the file mode of the dump
- *lvl* is the level of the dump (in the form *VvvRrrMm*, for example V05R01M0)

- *mode* is one of 'CP 32–bit', 'CP 64–bit', or 'NON-CP'
- *list* is a string of the Block Definition Library file names that are appropriate based on the characteristics of the dump.

The following is an example of the parameters passed to VMDTBPRF XEDIT. The values have been placed on separate lines for readability. The input to VMDTBPRF XEDIT contains all of this information in one line.

```
DUMPSPEC<2T0L997 DUMP0001 P1>
VER<V05R01M0>
MODE<CP 64-BIT>
BLKLIBS<HCQB8440 HCQB4440 HCQB4430 HCQB4420 HCQB4410 HCQB4310>
```

When the VMDTPRF XEDIT file exits, it *must* return a value containing the string BLKLIBS <*list*> with the list of Block Definition library file names that should be used by the VM Dump Tool. Other parameters that are returned will be ignored. The returned list will be searched in the order specified whenever a reference to a Block Definition Library is encountered.

Non-CP MAP User Exit

The VMDTNCPM VMDT macro is a non-CP MAP user exit. It runs during VM Dump Tool initialization and gathers module address and entry point information contained in a non-CP dump.

The VMDTMAP file provides the data needed to resolve symbols to addresses and addresses to symbols by the MAP subcommand and the EXTRACT MAPA and EXTRACT MAPN functions. This information is also used when you have issued SET REGS MAP to cause a number of VM Dump Tool subcommands to resolve addresses to symbols (examples are PSWS, GREGS and CPEBK). The VM Dump Tool contains enough knowledge of CP to extract the map information either from the CP symbol table in the dump, or by scanning CP storage in the dump and finding module prologs and epilogs. There are fewer specifics available for creating a VMDTMAP file for a non-CP dump because:

- The format of the input file can vary
- The load point in storage can vary from what is indicated in the load map
- CMS may or may not recognize where this program is loaded.

These unknowns are addressed by the non-CP MAP exit.

How the Non-CP MAP User Exit Runs

The non-CP MAP exit runs at the end of VM Dump Tool initialization (when needed) after subcommands and macros are available, before the user profile (VMDTPROF XEDIT) runs, and before the dump session screen is presented to the user.

During processing, the VM Dump Tool checks to whether a VMDTMAP file already exists for this dump. If one is found, no further action is taken. If one is not found, the VM Dump Tool tries to create one from the CP Symbol Table. If that fails, then it tries to scan CP storage in the dump looking for module prologs and epilogs to identify which modules and entry points are in the dump and at what addresses.

If this is not a CP dump, or if the VM Dump Tool can not create a VMDTMAP file by either of the above methods for any reason, it then looks for and runs a file

named VMDTNCPM VMDT (the non-CP MAP exit) . If no such file is found no further action is taken and map information is not available.

VMDTNCPM Macro Internal Logic

A sample VMDTNCPM VMDT macro is included with the VM Dump Tool. You can modify this macro to fit the needs of the software in the dump that you want to look at. It is recommended that you copy this file and not modify the original file that is shipped with the VM Dump Tool.

The sample VMDTNCPM VMDT macro does the following:

- The macro looks for a file called *filename* MAP *, where *filename* is the file name of the dump. If such a file is found, the macro then looks for a file called *filename* LOADMAP. If it finds either file type it assumes that the file is a load map and that it matches the dump. Then the macro reads in the file and extracts the map information.
- If the macro does not find a load map in the previous step, it asks the user to provide a new file name. If such a name is provided, it looks again for a file type of MAP or LOADMAP and extracts the map information. If no file name is provided, no map information will be available.
- The input file, *filname* MAP or *filename* LOADMAP file, can be in any of these three formats:
  - A load map from the CMS LOAD command with the FULLMAP option
  - A load map from the CMS LOAD command without the FULLMAP option
  - A load map from the output of the CP loader (which contains :READ lines, and other information).

The VMDTNCPM VMDT macro does not assume a load map format based on the specific file type. Any of the three formats above can be processed from a file type of either MAP or LOADMAP.

The macro must adjust addresses in the load map to reflect the actual addresses of the code in the dump. The VMDTNCPM macro obtains the value from label FRSTLOC in NUCON (at NUCON+56C), and suggests this to the user as a load point. The user can confirm that this is the correct address by hitting ENTER, or by supplying the address where the program described in the load map is actually loaded in the dump. Then the load map will be relocated to that address.

Inputs to VMDTNCPM

The only input to the VMDTNCPM macro is the file name of the dump, which is passed in as an argument.

Expected Output from VMDTNCPM

The VMDTNCPM VMDT macro creates a work file with a file name of VMDUMPTL and a file type of VMDTUT1. The work file has the following characteristics:

- A fixed record format with a record length of 16, which contains:
  - an 8-byte name field in EBCDIC, padded on the right with blanks (X'40')
  - A 4-byte address in binary
  - A 4-byte length field in binary. If this is a CSECT name, these 4 bytes are the length of the CSECT. If this is an entry point name, this field is binary zero.
- The data sorted in ascending order in the address field.

Upon successful return, the VM Dump Tool does one of the following:

- Copies the newly created file to the dump disk as *filename* VMDTMAP (if it has R/W access and there is space for it on that disk), or
- Renames the file to *filename* VMDTMAP on your A-disk.

Return Codes from VMDTNCPM

If the VMDTNCPM VMDT macro has created the required file without error, it returns a 0 return code and continues processing the map file. A non-zero return code indicates that it could not create the required map file. The VM Dump Tool continues running without the map information.

Usage Notes

1. The supplied VMDTNCPM VMDT macro functions correctly for the environments described, but has not been fully tested. It is provided as a sample only.
2. When taking a dump of Pipes or another program that is not loaded by normal CMS functions, you will have to obtain and supply the actual load point of the pipes (or other) code. Depending on the situation, this information may be available by using the NUCXMAP CMS command.
3. If the file VMDTNCPM VMDT is not found, VM Dump Tool initialization takes no further action, and no error is presented.
4. The VMDTNCPM VMDT macro is a normal VM Dump Tool macro. It can use VM Dump Tool, XEDIT, or REXX commands or functions.
5. Changes to a VMDTNCPM VMDT macro can be tested by invoking HCQINMAP VMDT, the VM Dump Tool map initialization macro. Use the FORCE parameter if a file name of VMDTMAP already exists but you wish to create another one. SET DEBUG ON provides additional information about the decisions that are made and any errors that can occur.

Appendix A. Cursor Directed Substitution Feature

You can specify one parameter of any subcommand in the VM Dump Tool indirectly by placing an underscore character (“_”) on the command line, placing the cursor on a value in the data area on the screen, and then pressing enter. The underscore will be replaced by the value pointed to by the cursor. For example, if you wanted to display x'100' bytes starting at 21D380, and ' 0021D380 ' was displayed on the screen, you could type 'd _.100' and move the cursor under the address. When you press enter, the command executed would be 'd 0021D380.100'.

This feature can prove to be even more useful when it is assigned to a PF key. For example, if you wanted to assign the above command to PF6, you could enter

```
set pf6 only macro HCQREAD VMDUMPTL d _.100
```

You could then place the cursor on an address, press PF6, and x'100' bytes of data starting at that address would be displayed.

When you use an underscore on the command line and the cursor is not in the data area of the screen, a value of zero (“0”) is used for substitution.

Appendix B. Writing Macros for the VM Dump Tool

The subcommands and macros included with the VM Dump Tool help you to analyze dump data interactively, and in most situations there is a subcommand or macro that will help you locate the data you need in a usable format. However, you can write your own macros to customize, automate, and expand the functions of the VM Dump Tool, 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
  - Simpler output (such as changing technical jargon to English)
  - Additional annotations
  - Additional time/date/tracking information.
- Automate repetitive tasks.

This section explains how to write macros which use VM Dump Tool subcommands. Before writing your own macro 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.

What Is a VMDUMPTL Macro?

A VM Dump Tool macro is a file you invoke from the VMDUMPTL environment. This environment exists whenever VM Dump Tool is being used. VM Dump Tool macros can have a file type of VMDT, XEDIT, or SCAN. A macro is invoked from the command line as any other VM Dump Tool subcommand would be, or may be invoked from another macro.

A macro can be executed by entering only its name, its name and any parameters needed for its execution, or you can invoke the macro using a function key.

A macro file can invoke:

- VM Dump Tool subcommands
- VMDT macros
- XEDIT subcommands
- REXX instructions
- Calls to other EXECs or modules
- CMS and CP commands.

Creating a Macro File

A macro is a 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:

- The file name may be any file name that is acceptable to CMS
- The file type must be VMDT, XEDIT, or SCAN

- The file mode can be any disk or directory to which you have write access (usually your A-disk).

Using VM Dump Tool Subcommands in a Macro

A macro can invoke any VM Dump Tool subcommand. Most subcommands look for specific data in the dump, format it, and queue it to be written to the session log. Also, your macro can read the output of subcommands or other macros and modify or enhance the output as needed before displaying it in the dump session.

While any VM Dump Tool subcommand or macro may be invoked from a VM Dump Tool macro, only a subset of these are considered to be actual programming interfaces (noted with each subcommand or macro).

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 REXX instructions, they are passed to the specified (or default) 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 VMDUMPTL causes the line to be passed to VMDUMPTL.
- ADDRESS XEDIT causes the line to be passed to XEDIT.
- ADDRESS CMS or ADDRESS COMMAND causes the line to be passed to CMS.
- A term by itself causes the line to be passed to the default environment.

When you use VMDT as the file type for your macro, VMDUMPTL is the default environment. When you use XEDIT as the file type for your macro, XEDIT is the default environment. When you use SCAN as the file type for your macro, SCAN is the default environment.

Note that the preferred file type for VM Dump Tool macros is VMDT. Starting with version 5, release 2.0, all of the IBM-supplied VM Dump Tool macros are renamed from their former file type of XEDIT to VMDT. Any customer-written VM Dump Tool macro which invokes an IBM-supplied VM Dump Tool macro using **ADDRESS XEDIT** will have to be changed to use **ADDRESS VMDUMPTL**. Customer-written VM Dump Tool macros with a file type of XEDIT will continue to function as coded (no changes are needed). Also, customer-written VM Dump Tool macros which invoke other customer-written macros with a file type of XEDIT will continue to function as coded.

Sending Data to the Dump Session

The CMS program stack is the interface that allows your macro to place information in the dump session. Your macro can add output to the program stack with any combination of the REXX QUEUE and PUSH instructions, the Pipes STACK stage or the CMSSTACK assembler language macro.

Note that other programs expect to be able to use the program stack as well. If your macro invokes another VM Dump Tool macro, the called macro will also

leave its output in the program stack. Your macro can simply leave it there (so it will go to the dump session), or read, process, or modify it, and then place it back into the program stack.

If your macro can be called from other macros, it is usually prudent to save the contents of the program stack at entry, restore it to its original content when your macro is ready to exit, and then add the output that your macro generates.

Sample Macros

Example 1 — Using QUEUE: The following macro displays one line of output to the dump session as well as the input parameters that were passed in:

```
/* SAMPLE1 - generate a line of output */
Queue 'hello VM Dump Tool user'

Arg inputs
Queue 'inputs received >'inputs'<'

Sample macros:

>>> sample1
hello VM Dump Tool user
inputs received ><

>>> sample1 some input parms
hello VM Dump Tool user
inputs received >SOME INPUT PARMS<
```

Example 2 — Using PIPES: The following macro demonstrates how to read the output from another command (or macro, as in this case), and how to send PIPES output to the dump session.

```
/* SAMPLE2 VMDT - output from pipes */
'VMDBKS'
Address COMMAND 'PIPE',
   '|   stack',              /* get the contents of the stack */
   '|   locate w2 /base/',   /* keep only 'base' type users */
   '|   locate w1 /SYS/',    /* find all userids w/SYS in them */
   '|   buffer',             /* hold output until all is read */
   '|   stack'               /* to dump session */
```

The macro invokes the VMDBKS macro, which places its output in the program stack. This macro picks up that output with the PIPES STACK stage, proceeds to filter the data with two LOCATEs, and then places its output back into the program stack.

Since PIPES is both reading from the stack and writing to it, it is possible for the data to get confused. For this reason, the BUFFER stage is included. BUFFER accumulates all the data until the end of the input data stream is reached. It then lets all of the data proceed. With the BUFFER, we know that all of the output from VMDBKS has been processed before the new output is placed back in the program stack.

Example 3 — Preserving the Program Stack: If you want to use the program stack and don't know what is in it, you need to save and restore its contents. This can be done as shown in the following example:

```
/* SAMPLE3 VMDT - get CPUID, save the program stack */

Address COMMAND 'PIPE',          /* save incoming stack contents */
   '|   stack',                  /* get the current stack */
   '|   stem prvstk.'            /* save it in this stem */
```

```
'CPU'                           /* issue CPUID command */

Address COMMAND 'PIPE',
   '|   stack',                 /* read output of CPUID */
   '|   buffer',                /* chuck all the rest */
   '|   take 1',                /* keep just the first line */
   '|   spec /cpu id/ 1 w3-4 nw', /* keep just these words */
   '|   var keepit'             /* to this variable */

Address COMMAND 'PIPE',
   '|   preface stem prvstk.',  /* get incoming stack contents */
   '|   append var keepit',     /* add our output */
   '|   stack'                  /* to the dump session */
```

The output from this macro is the following:

```
>>> sample3
cpu id 00131515 20640000
```

(To prove that this program is working correctly, you could write another EXEC, have it stack something, then invoke SAMPLE3 and verify that the output stacked before invoking SAMPLE3 is placed in the program stack in the correct order.)

Displaying Errors on the VM Dump Tool Screen

You can use the following functions to issue prompts and messages from the macro to the VM Dump Tool screen. Note that data that is placed in the program stack is added to the VM Dump Tool session.

MSG An XEDIT subcommand that displays a message on the message line.

EMSG
An XEDIT subcommand that displays a message on the message line and sounds the alarm.

CMSG
An XEDIT subcommand that displays a message on the command line.

SAY A REXX statement that displays a message on the virtual machine console.

QUEUE
A REXX statement that displays a message in the VM Dump Tool session by placing it in the program stack in FIFO order.

PUSH A REXX statement that displays a message in the VM Dump Tool session by placing it in the program stack in LIFO order.

CP MSG
A CP command that displays a message on the virtual machine console (outside the VM Dump Tool session).

Preserving and Restoring Prefixing

There are cases where you may want to have prefixing on, and others where you may want to have it off. For example, if your macro will follow the PFXPG chain, you probably want prefixing set off so all of the PFXPGs can be referenced in the same way. Alternatively, if your macro tries to read a field out of the PFXPG, it is a lot easier to have prefixing turned on and simply reference an address in low storage. In both cases, the macro should restore the prefixing state that existed before it was invoked.

Use the PREFIX subcommand with OFF or ON and the RCONLY option to set the prefixing state and return the old value. This old value can be used later to restore the previous environment.

Example — Setting Prefixing On: This example saves the old setting and sets prefixing on, then restores it at the end.

```
/* SAMPLE4 VMDT - save & restore PREFIXing */

'PREFIX ON RCONLY'         /* set it on, return old value */
pfxorig = rc               /* save in a local variable */

Queue 'SAMPLE4 macro, prefixing was set to' pfxorig

/* at exit: */
'PREFIX' pfxorig           /* restore previous setting */
```

The following is output from this macro showing that the setting has been properly restored:

```
>>> prefix
Prefixing of Page 0 is on
>>> sample4
SAMPLE4 macro, prefixing was set to 1
>>> prefix
Prefixing of Page 0 is on

>>> prefix off
Prefixing of Page 0 is off
>>> sample4
SAMPLE4 macro, prefixing was set to 0
>>> prefix
Prefixing of Page 0 is off
```

Reading Data from Storage

Most macros will want to read values out of storage for analysis or display. There are a number of ways to do this and present the data. The most direct is the SETVAR subcommand. An alternative is with the BLOCK macro, either letting it produce the output or having it return the contents of a field. All of these are shown in the following example.

Example — Reading Data with SETVAR and BLOCK:

```
/* SAMPLE5 VMDT - read data with SETVAR and BLOCK */

'PREFIX ON RCONLY'             /* set prefix on, get old value */
pfxorig = rc                   /* save former setting */

'SETVAR WORD syscmad 998'      /* read from storage, @ of SYSCM */
Queue 'SYSCM is at' syscmad    /* msg with that info */

'BLOCK SYSCM' syscmad 'FIELD SYSOPER'        /* get BLOCK info */

'BLOCK SYSCM' syscmad 'SETVAR FIELD SYSOPER' /* get field contents */
Queue 'operator userid is' sysoper           /* to the output */

'PREFIX' pfxorig               /* restore PREFIX setting */
```

The output from this macro is the following:

```
>>> sample5
SYSCM is at 00026000
+0068 SYSOPER        'OPERATOR'         USERID OF PRIMARY SYSTEM
                                        OPERATOR
operator userid is OPERATOR
```

Note the following:

- You can use the SETVAR subcommand to read out registers for the failing processor.
- You can use the DISPLAY subcommand to display storage.
- The BLOCK macro has other parameters which allow you to display a range of displacements in a control block, or display a length from a displacement in a control block.

Writing Macros

HCQGDSPL Macro: In order for a macro to use data from a field of a control block, the macro must know the displacement of that field. But sometimes the displacements can change over time, usually on a release boundary, but sometimes with an APAR fix.

For this reason, VM Dump Tool macros with hard-wired displacements can get out of date and no longer function correctly. When this happens it is necessary to debug the failing macro to find what changed and change the macro to match.

A new macro, HCQGDSPL EXEC, is provided which obtains displacement information from the BLOCK data (see the VM Dump Tool BLOCK subcommand). HCQGDSPL is not intended to be invoked from the command line. It must be invoked as a function from another macro.

The following is an illustration of this problem. The goal is to display the value of VMDRTERM for each VMDBK in a dump. VMDRTERM is at displacement 558 in the VMDBK, so this value is simply hard coded into the macro.

```
/* */
Address COMMAND 'PIPE',           /* Invoke pipes */
   '|   avmdbks',                 /* get list of all VMDBKs */
   '|   getword 558 w1',          /* get VMDRTERM for each */
   '|   buffer',                  /* accumulate all output */
   '|   stack'                    /* add to the dump session */
```

The output from this macro is the following:

```
>>> play
00002000 00000000
01236000 012DEB38
04789000 00000000
0606D000 00000000
04786000 00000000
04704000 00000000
0453C000 00000000
04526000 00000000
04506000 00000000
```

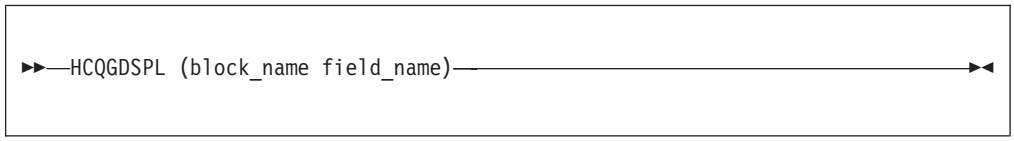
VMDRTERM is the address of the Real DEVice BlocK for the console device. Zero indicates that the virtual machine is running disconnected. The output above says that only the second user listed actually has a terminal.

The problem, of course, is that if the field moves, we're no longer looking at the VMDRTERM data and the macro will be gathering data or making decisions based on faulty information.

With a module, you can often recompile or reassemble it and fix the problem. That doesn't work here. First you have to figure out that this macro has a dependency on this field displacement, that the displacement has changed, figure out what the new value is, then edit the macro and change it to the new value. If the macro supports multiple releases, further logic is needed to figure out which release is present and which corresponding value to use.

The BLOCK macro accomplishes this by varying the list of active BLOCK libraries according to the release of the dump you're looking at. The later BLOCK libraries are 'deltas' on a base file. When searched in order from newest release to oldest, the most current one will be the first one found, and relevant to the dump at hand. It turns out that BLOCK calls another service under the covers to do this, and it's fairly easy to 'borrow' the same service, which is what the HCQGDSPL macro does.

Syntax:



Where:

block_name
The name of a control block (eg VMDBK, ASCBK, CPVOL). This can be specified as upper case or lower case.

field_name
The name of the desired field within that block.

Note:

1. Note that these are separated by a blank, not a comma.
2. If the macro encounters a problem, it returns a negative return code.

Output: HCQGDSPL returns the 4-digit hex value of the displacement of that field.

Example: Here's the same example from above using HCQGDSPL.

```
/* */
@VMDRTERM = HCQGDSPL('vmdbk' 'vmdrterm') /* get field displacement */

Address COMMAND 'PIPE',           /* invoke pipes */
   '|   avmdbks',                 /* get list of all VMDBKs */
   '|   getword' @VMDRTERM 'w1',  /* get VMDRTERM for each */
   '|   buffer',                  /* collect them all */
   '|   stack'                    /* to the dump session */
```

There's nothing magic about the string '@VMDRTERM'. This simply uses a convention of field name, in caps, prefixed by @ for field name variables. You can call it anything you'd like, but this could make it easier to remember the variable name for a field displacement.

Of course the output looks the same as before. But now we know that this macro is much more insulated from changes in a new release.

VM Dump Tool Search Order

The following is the search order VM Dump Tool uses:

1. VMDT macro (see Usage Note 1)
2. VM Dump Tool subcommand (see Usage Note 1)
3. XEDIT macro
4. SCAN macro
5. XEDIT commands, CMS commands, CP commands, and EXECs, depending on your XEDIT settings.

Usage Notes for Writing Macros

1. When the input is from the command line, the command handler first searches for a VMDT macro and then for a VM Dump Tool subcommand. This allows you to override a VM Dump Tool subcommand in order to provide enhanced output or to circumvent a problem.

   When the input is from a macro, the command handler first searches for a VM Dump Tool subcommand and then for a VMDT macro. This allows a macro to expect predictable output even if a VM Dump Tool subcommand is overridden.
2. When a string you enter is not recognized, the following message is issued by XEDIT:

   ```
   DMSXDC542E  No such subcommand:   (subcommand_name)
   ```
3. The stack will have been cleared when the macro gains control, so parameters for the macro should not be stacked.
4. The default addressing environment in a VM Dump Tool macro is VMDUMPTL.
5. XEDIT macros can also be invoked from within a VM Dump Tool macro by prefixing with **ADDRESS XEDIT**.
6. Some subcommands that can be used in macros are CPUID, DUMPNAME, DUMPTYPE, EXTRACT, PREFIX, SETVAR, and VMDBK.
7. The VM Dump Tool subcommands CP and CMS can be used to send commands directly to those command environments.
8. Output from VM Dump Tool subcommands is stacked FIFO. This output may be read by the macro with PULL or PIPE STACK, or left in the stack to be displayed as described below. In general the macro may use the stack as it sees fit.
9. Anything in the stack when the macro exits will be added to the dump session as output from the macro, just as it would be if the output were from a VM Dump Tool command. The stack is the primary means for a macro to pass back its output to the user.
10. Starting with version 5, release 2.0, all of the IBM-supplied VM Dump Tool macros are renamed from their current file type XEDIT to VMDT. Any customer-written VM Dump Tool macro which invokes an IBM-supplied VM Dump Tool macro using **ADDRESS XEDIT** will have to be changed to use **ADDRESS VMDUMPTL**

About SET DEBUG

Note: The VMDTSET DEBUG subcommand is preferred for new code over SET DEBUG, but SET DEBUG will continue to work.

The SET DEBUG command allows you to perform some debugging in a VM Dump Tool macro without having to change the macro itself. When DEBUG is set to ON the following occurs:

- A flag is set. This flag is available to any macro via the return code from the QUERY DEBUG command.
- All subsequent error messages produced by VM Dump Tool subcommands are written to the virtual machine console.

If you have your own user-written macros, you can modify them with a small amount of code so that the flow of control from one macro to another is displayed. Also, you can display non-zero codes returned to the macro. While this new code would be a permanent addition to the macro, you can turn it on and off using SET DEBUG. By writing additional code, you can display more detailed information when DEBUG is ON.

For more information, see "Using SET DEBUG" on page 208.

VMDUMPTL Macro Examples

This section shows you how to create a new VMDUMPTL macro function.

Creating a New VM Dump Tool Macro Function

The following examples show you how to use a combination of XEDIT subcommands and VM Dump Tool subcommands to create a new VM Dump Tool macro function. Note that a VMDT macro can be used to override a VM Dump Tool subcommand or an XEDIT macro. For more information, see "VM Dump Tool Search Order" on page 202.

In Figure 14 on page 204, a new VM Dump Tool macro displays 16 bytes from a specified offset within the prefix page of each CPU. To get the same information using VM Dump Tool subcommands from the command line would require you to enter the CPUID subcommand, write down the addresses of the prefix pages, add the offset to each address, and then, for each address, enter the DISPLAY subcommand to display the data.

```
001 /*---------------------------------------------------------------*/
002 /* SAMPA VMDT - Sample VM Dump Tool Macro.                       */
003 /*                                                               */
004 /* Display 16 bytes from the specified offset from the prefix    */
005 /* page of each CPU.                                             */
006 /*                                                               */
007 /*---------------------------------------------------------------*/
008 Address VMDUMPTL               /* default: send to VM Dump Tool */
009 Numeric Digits 12               /* Handle 8-digit hex numbers    */
010
011 /*---------------------------------------------------------------*/
012 /* Handle the case where the offset is not specified             */
013 /*---------------------------------------------------------------*/
014 Parse Arg offset .                /* get offset from command line */
015 If offset = '' Then Do           /* if nothing there... */
016    Queue 'Syntax:  SAMPA offset'
017    Queue '            where offset is the offset into the'
018    Queue '            failing Prefix Pages to display'
019    Exit 1                        /* exit with an error */
020    End
021
```

Figure 14. Example of a VM Dump Tool Macro (Part 1 of 2)

```
022 /*---------------------------------------------------------------*/
023 /* Be sure the offset is valid hex                               */
024 /*---------------------------------------------------------------*/
025 If Datatype(offset,'X') <> 1 Then Do
026    Queue 'Offset contains non-hex data: ' offset
027    Exit 2
028    End
029
030 /*---------------------------------------------------------------*/
031 /* Be sure the offset is not too large                           */
032 /*---------------------------------------------------------------*/
033 max = 4096 - 16               /* leave room to display 16 bytes */
034 If X2D(offset) > max Then Do   /* if bigger... */
035    Queue 'offset too large,' D2X(max) 'is max'
036    Exit 3                     /* exit with error */
037    End
038
039 /*---------------------------------------------------------------*/
040 /* Gather the Prefix Page data                                   */
041 /*---------------------------------------------------------------*/
042 'CPUID'                         /* get CPUID information   */
043 Address COMMAND 'PIPE',         /* take the output apart */
044    '|   stack',                 /* read from the program stack  */
045    '|   drop 2',                /* drop non-relevant     */
046    '|   spec w8 1',             /* get PFXPG address     */
047    '|   stem pfxpgs.'           /* results to an array   */
048
049 'prefix off rconly'             /* to OFF, get previous setting */
050 oldprefix = rc                  /* save to restore later */
051
052 /*---------------------------------------------------------------*/
053 /* Display the results                                           */
054 /*---------------------------------------------------------------*/
055 Do i = 1 to pfxpgs.0            /* do for each Prefix Page      */
056    'display' XADD(pfxpgs.i offset)'.10'  /* display 16 bytes     */
057    End                          /* end of loop                  */
058
059 'prefix' oldprefix              /* restore old PREFIX setting   */
060
061 Exit 0                          /* everything is beautiful...   */
062
063 /*---------------------------------------------------------------*/
064 /* XADD Subroutine - Add 2 hex numbers to a hex result           */
065 /*---------------------------------------------------------------*/
066 XADD:                          /* Hex ADD */
067    Parse Arg h1 h2 .           /* h1 is Hex, h2 is Hex */
068    Return D2X((X2D(h1) + X2D(h2)), 8)
```

Figure 15. Example of a VM Dump Tool Macro (Part 2 of 2)

A description of how the macro works follows:

```
001 /*---------------------------------------------------------------*/
002 /* SAMPA VMDT - Sample VM Dump Tool Macro.                       */
003 /*                                                               */
004 /* Display 16 bytes from the specified offset from the           */
005 /* prefix page of each CPUs                                      */
006 /*                                                               */
007 /*---------------------------------------------------------------*/
```

Comments. The first line must be a comment (required by REXX).

```
008 Address VMDUMPTL           /* default: send to VM Dump Tool */
```

The default environment, if not otherwise specified, is to send subcommands to the VM Dump Tool. This is optional. The default in a macro with a file type of VMDT is VMDUMPTL.

```
009 Numeric Digits 12          /* Handle 8-digit hex numbers    */
```

The addresses we are dealing with can be larger than the maximum number normally handled by REXX. Make sure that we can handle an address value.

```
010
011 /*-------------------------------------------------------------*/
012 /* Handle the case where the offset is not specified           */
013 /*-------------------------------------------------------------*/
014 Parse Arg offset .             /* get offset from command line   */
```

Take the first operand and assign it to the variable *offset*. The period says to ignore any other operands.

```
015 If offset = '' Then Do           /* if nothing there... */
```

Check for the required operand. If it was not entered then execute the following section of code.

```
016    Queue 'Syntax:  SAMPA offset'
```

Put this and the following lines in the program stack where they will be placed in the dump session when the macro exits.

```
017    Queue '            where offset is the offset into the'
018    Queue '            failing Prefix Pages to display'
019    Exit 1                          /* exit with an error */
```

Return an error to the caller.

```
020    End
```

End of the group of statements to execute when offset is null.

```
021
022 /*-------------------------------------------------------------*/
023 /* Be sure the offset is valid hex                             */
024 /*-------------------------------------------------------------*/
025 If Datatype(offset,'X') <> 1 Then Do
```

See if the variable *offset* contains valid hex characters.

```
026    Queue 'Offset contains non-hex data: ' offset
```

If not then present an error message to that effect.

```
027    Exit 2
```

Exit back to the user with a unique error code.

```
028    End
029
030 /*-------------------------------------------------------------*/
031 /* Be sure the offset is not too large                         */
032 /*-------------------------------------------------------------*/
```

Because we want to display 16 bytes of data, we want to be sure that we do not go off the end of the Prefix Page.

```
033 max = 4096 - 16            /* leave room to display 16 bytes */
```

This is highest address from which we can display 16 bytes in the page.

```
034 If X2D(offset) > max Then Do       /* if bigger...   */
```

If the offset given us is too large, it is an error.

```
035    Queue 'offset too large,' D2X(max) 'is max'
```

```
036    Exit 3                          /* exit with error */
037    End
038
039 /*-------------------------------------------------------------*/
040 /* Gather the Prefix Page data                                 */
041 /*-------------------------------------------------------------*/
042 'CPUID'                              /* get CPUID information */
```

Issue the VM Dump Tool CPUID subcommand. The output is of the form:

```
CPUID = 75016452 96720000

  CPU address is 0000    Prefix register is 04508000 (failing)
  CPU address is 0001    Prefix register is 76574000
  CPU address is 0002    Prefix register is 765B0000
  CPU address is 0003    Prefix register is 765EC000
  CPU address is 0004    Prefix register is 7665A000
```

```
043 Address COMMAND 'PIPE',            /* take the output apart */
```

Invoke Pipelines to extract the desired information.

```
044    '|   stack',                    /* read from the program stack */
```

Read all of the lines produced by the CPU subcommand above.

```
045    '|   drop 2',                    /* drop non-relevant */
```

Drop the first 2 lines: CPUID line and blank line.

```
046    '|   spec w8 1',                 /* get PFXPG address */
```

Pick off just the 8th word, the Prefix Page address.

```
047    '|   stem pfxpgs.'          /* results to an array */
```

Save the result in the stem variable called PFXPGS.

```
048
049 'prefix off rconly'    /* to OFF, get previous setting */
```

Prefixing defaults to ON for the Prefix Page of the failing processor. Set Prefixing off, return the previous state as the return code. We do this so we can display from all prefix pages the same way. Without this, we'd have to add the displacement to zero, instead of to the prefix page address, to display from the prefix page of the failing processor.

```
050 oldprefix = rc        /* Other commands (such as DISPLAY) also */
                          /* set the return code.  We save it here so  */
                          /* we can restore it later in line 59. */
```

Save the previous state so we can restore it later.

```
051
052 /*-------------------------------------------------------------*/
053 /* Display the results                                         */
054 /*-------------------------------------------------------------*/
055 Do i = 1 to pfxpgs.0                /* do for each Prefix Page */
```

Call XADD to add the offset in the page to the address of the prefix page for this CPU.

```
056    'display' XADD(pfxpgs.i offset)'.10'     /* display 16 bytes */
```

Invoke the VM Dump Tool subcommand to display the data.

```
057    End                             /* end of loop */
058
059 'prefix' oldprefix    /* restore the former PREFIX setting */
```

Restore the former prefixing setting that was obtained above.

```
060
061 Exit 0            /* everything is beautiful... */
```

Success, go back to the user.

```
062
063 /*-------------------------------------------------------------*/
064 /* XADD Subroutine - Add 2 hex numbers to a hex result         */
065 /*-------------------------------------------------------------*/
```

XADD is a small subroutine to take the complexity of hex addition out of the mainline of the program.

```
066 XADD:                                      /* Hex ADD */
```

The name of the subroutine: XADD.

```
067 Parse Arg h1 h2 .               /* h1 is Hex, h2 is Hex */
```

Get the 2 hex input values to add together.

```
068 Return D2X((X2D(h1) + X2D(h2)), 8)
```

Convert each to decimal, add together, convert result to hex, pad the result to eight characters, and return to the caller.

Macro Examples:

```
>>> sampa
Syntax:  SAMPA offset
            where offset is the offset into the failing
            Prefix Page to display

>>> sampa asdf
Offset contains non-hex data:  asdf

>>> sampa 0
00000000_04508000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*
00000000_76574000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*
00000000_765B0000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*
00000000_765EC000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*
00000000_7665A000 00FC3000 8467B4F8 FFFFFFFF FFFFFFFF *....d..8........*

>>> sampa 90
00000000_04508090 01F65000 000F0000 00000000 047479CA *.6&..........`.*
00000000_76574090 00EBB000 000F0000 00000000 04666178 *.............../.*
00000000_765B0090 00E9F000 000F0000 00000000 0464B90E *.Z0.............*
00000000_765EC090 06987001 000F0000 00000000 04666178 *.q............/.*
00000000_7665A090 00020000 000F0000 00000000 0464B90E *................*
```

Using SET DEBUG

Note: The VMDTSET DEBUG subcommand is preferred for new code over SET DEBUG, but SET DEBUG will continue to work.

Example 1 – Basic SET DEBUG Function: This sample code performs testing to see if SET DEBUG is enabled, shows that the macro was entered, and enables the tracking of non-zero return codes.

```
    /*-----------------------------------------------------*/
    /* MyMacro1 VMDT - sample to show use of SET DEBUG     */
    /*-----------------------------------------------------*/

001  Parse Source . . macname .
```

```
    /*-------------------------------------------------------*/
    /*  This gets the name of the macro so it doesn't have */
    /*  to be entered specifically.  This is used later to */
    /*  display the name of the macro in lines 5 and 11.   */
    /*-------------------------------------------------------*/

    /*-------------------------------------------------------*/
    /* Set up environment if in DEBUG mode                   */
    /*-------------------------------------------------------*/
002  'QUERY DEBUG RCONLY'
    /*-------------------------------------------------------*/
    /*  This issues the QUERY DEBUG command to the VM Dump */
    /*  Tool. RCONLY specifies that only the return code   */
    /*  is displayed.                                        */
    /*-------------------------------------------------------*/
003  debugrc = rc
    /*-------------------------------------------------------*/
    /*  Since the variable named rc is set at exit from    */
    /*  any invocation, we need to copy its value here to  */
    /*  a local variable called arbitrarily called debugrc */
    /*  so it will be available later.                     */
    /*-------------------------------------------------------*/
004  If debugrc = 1 Then Do
    /*-------------------------------------------------------*/
    /*  This tests whether DEBUG was actually set ON.      */
    /*-------------------------------------------------------*/
005     Say '---> Entering' macname
    /*-------------------------------------------------------*/
    /*  If DEBUG is ON, this provides the tracing          */
    /*  information to tell the user that this macro was   */
    /*  entered                                              */
    /*-------------------------------------------------------*/
006     Trace 'E'
    /*-------------------------------------------------------*/
    /*  This tells CMS to display any function invocation  */
    /*  which gets a non-zero return code.                 */
    /*-------------------------------------------------------*/
007     End
    /*-------------------------------------------------------*/
    /*  This is the end of the block for the test          */
    /*  IF DEBUGRC = 1.                                      */
    /*-------------------------------------------------------*/

    /*-------------------------------------------------------*/
    /* The macro logic                                       */
    /*-------------------------------------------------------*/
008  Arg addr .
    /*-------------------------------------------------------*/
    /*  This gets the parameter specified on the           */
    /*  invocation and assigns it to the variable called   */
    /*  addr.                                                */
    /*-------------------------------------------------------*/

009  'SETVAR BYTE stgt' addr
    /*-------------------------------------------------------*/
    /*  This is the main processing of the macro.  It      */
    /*  reads the storage from the specified location into */
    /*  a variable called addr.                              */
    /*-------------------------------------------------------*/
010  Queue 'Storage at location' addr 'contains' stg
    /*-------------------------------------------------------*/
    /* This sends the result to the dump session.          */
    /*-------------------------------------------------------*/

    /*-------------------------------------------------------*/
    /* Exit Routine                                          */
    /*-------------------------------------------------------*/
```

```
011  If debugrc = 1 Then Say '<--- Leaving ' macname
    /*-----------------------------------------------------*/
    /*  This provides another trace to the virtual machine */
    /*  console when the macro exits.                      */
    /*-----------------------------------------------------*/
012 Exit 0
```

When this macro is invoked with an address, the following is added to the dump session:

```
>>> mymacro1 4
Storage at location 4 contains FF
```

When DEBUG is OFF, nothing is displayed on the virtual machine console.

When DEBUG is ON, the same data as above is added to the dump session:

```
>>> set debug on
   complete
   >>> mymacro1 4
   Storage at location 4 contains FF
```

The following is also displayed on the virtual machine console:

```
---> Entering MYMACRO1
<--- Leaving  MYMACRO1
```

When DEBUG is OFF, when this macro is invoked with no inputs the following is displayed in the dump session:

```
>>> mymacro1
   HCQSTV002E Address missing
   Storage at location 4 contains STG
```

It is not very clear just what the problem is with only the above output to go by. However, when DEBUG is ON, the following additional information is displayed on the virtual machine console:

```
>>> set debug on
   complete
   ---> Entering MYMACRO1
   HCQSTV002E Address missing
       22 *-* 'SETVAR BYTE stg' addr
          +++ RC(2) +++
   <--- Leaving  MYMACRO1
```

What has happened is that the macro expects an input address but none was specified, so SETVAR didn't get all of the required inputs. As you can see, when DEBUG is ON it is easier to see exactly where the problem occurred.

Example 2 – Displaying Internal Information: You can add code to a macro so that it displays information internal to the macro when DEBUG is ON. The following is an example of this. Note that Lines 9 and 11 use the value of the DEBUG return code to display (or not) other information.

```
    /*-----------------------------------------------------*/
    /* MyMacro2 VMDT - sample to show use of SET DEBUG     */
    /*-----------------------------------------------------*/
001  Parse Source . . macname .

    /*-----------------------------------------------------*/
    /* Set up environment if in DEBUG mode                 */
    /*-----------------------------------------------------*/
002  'QUERY DEBUG RCONLY'
003  debugrc = rc
004  If debugrc = 1 Then Do
```

```
005     Say '---> Entering' macname
006     Trace 'E'
007     End

    /*-------------------------------------------------------*/
    /* The macro logic                                       */
    /*-------------------------------------------------------*/
008  Arg addr .
009  If debugrc = 1 Then Say 'input addr >'addr'<'
010  'SETVAR BYTE stg' addr
011  If debugrc = 1 Then Say 'stg was set to' stg', rc='rc
012  Queue 'Storage at location' addr contains' stg

    /*-------------------------------------------------------*/
    /* Exit Routine                                          */
    /*-------------------------------------------------------*/
013  If debugrc = 1 Then Say '<--- Leaving ' macname
```

When the macro is run with an address, we get the a correct result:

```
>>> mymacro2 4
Storage at location 4 contains FF
```

When DEBUG is ON, the following is displayed on the virtual machine console. This helps you see what the macro was actually doing.

```
---> Entering MYMACRO2
input addr >4<
stg was set to FF, rc=0
<--- Leaving  MYMACRO2
```

When an input address is not specified, the following information is displayed:

```
---> Entering MYMACRO2
input addr ><
HCQSTV002E Address missing
    24 *-* 'SETVAR BYTE stg' addr
       +++ RC(2) +++
stg was set to STG, rc=2
<--- Leaving  MYMACRO2
```

No address was specified, so *addr* has a null value.

Example 3 – Using a Subroutine to Display Debug Information: The following macro uses a subroutine to check whether DEBUG is ON.

```
     *-------------------------------------------------------*/
    /* MyMacro3 VMDT - sample to show use of SET DEBUG       */
    /*-------------------------------------------------------*/
001  Parse Source . . macname .

    /*-------------------------------------------------------*/
    /* Set up environment if in DEBUG mode                   */
    /*-------------------------------------------------------*/
002  'QUERY DEBUG RCONLY'
003  debugrc = rc
004  If debugrc = 1 Then Do
005     Say '---> Entering' macname
006     Trace 'E'
007     End

    /*-------------------------------------------------------*/
    /* The logic of the macro                                */
    /*-------------------------------------------------------*/
008  Arg addr .
009  Call DEBUG 'input addr >'addr'<'
010  'SETVAR BYTE stg' addr
```

```
011  Call DEBUG 'stg was set to' stg', rc='rc
012  Queue 'Storage at location' addr 'contains' stg

    /*-----------------------------------------------------*/
    /* Exit Routine                                        */
    /*-----------------------------------------------------*/
013  If debugrc = 1 Then Say '<--- Leaving ' macname
014  Exit 0

    /*-----------------------------------------------------*/
    /* DEBUG subroutine - display info if in debug mode    */
    /*-----------------------------------------------------*/
015  DEBUG:
016     If debugrc <> 1 Then Return
017     Parse Arg debugin
018     Say debugin
019     Return
```

Lines 9 and 11 call the subroutine shown in lines 15-19 instead of testing the variable directly. This simplifies the main line code by putting the local test for DEBUG mode and the display function in one place.

Example 4 – A More Robust Macro: The following macro adds more function than the previous examples. Note that you can add more to this macro, such as checking that all of the characters in the address are valid hexadecimal characters, and that the address is not too large to be an address. (Use EXTRACT HEX64 or EXTRACT HEX31 to do this.)

```
    /*-----------------------------------------------------*/
    /* MyMacro4 VMDT - sample to show use of SET DEBUG     */
    /*-----------------------------------------------------*/
001  Parse Source . . macname .

    /*-----------------------------------------------------*/
    /* Set up environment if in DEBUG mode                 */
    /*-----------------------------------------------------*/
002  'QUERY DEBUG RCONLY'
003  debugrc = rc
004  If debugrc = 1 Then Do
005     Say '---> Entering' macname
006     Trace 'E'
007     End

    /*-----------------------------------------------------*/
    /* The logic of the macro                              */
    /*-----------------------------------------------------*/
008  Arg addr .
009  Call DEBUG 'input addr >'addr'<'

010  If addr = '' Then Do
011     Queue 'an input address must be specified'
012     Exit 99
013     End

014  'SETVAR BYTE stg' addr
015  svrc = rc
016  Call DEBUG 'stg was set to' stg', rc='rc

017  If svrc <> 0 Then Do
018     Queue 'rc' svrc 'received from SETVAR in reading storage'
019     Exit 98
020     End

021  Queue 'Storage at location' addr 'contains' stg

    /*-----------------------------------------------------*/
```

```
    /* Exit Routine                                          */
    /*------------------------------------------------------*/
022  If debugrc = 1 Then Say '<--- Leaving ' macname
023  Exit 0

    /*------------------------------------------------------*/
    /* DEBUG subroutine - display info if in debug mode     */
    /*------------------------------------------------------*/
024  DEBUG:
025     If debugrc <> 1 Then Return
026     Parse Arg debugin
027     Say debugin
028     Return
```

Line 10 is a new line of code (from the previous examples) that checks to see if the required input was specified. If it was not, line 11 displays an error message to the dump session, and line 12 exits with an error code. Another new test has been added at line 17 to see if the SETVAR was successful. If it was not, this logic will issue a message and then exit. Note that the return code from the SETVAR at line 14 is saved around the call to DEBUG.

Example 5 – Working with Non-Zero Return Codes: Non-zero return codes do not always indicate an error. This example helps you determine whether a non-zero return code is an error by giving you more information about the return code. Suppose the following information is displayed from the CALLERS macro:

```
---> Entering CALLERS
       39 *-* 'DUMPTYPE RCONLY'
          +++ RC(1) +++
       91 *-* 'PREFIX ON  RCONLY'
          +++ RC(1) +++
   <--- Leaving  CALLERS
```

The return code of 1 from DUMPTYPE indicates that we are looking at a CP 64-bit dump. The return code of 1 from PREFIX indicates that PREFIX was already set on. Note that neither of these return codes indicates an error.

Appendix C. VM Dump Tool Comparison with Dump Viewing Facility

The following tables are a high-level comparison of the functions available from the Dump Viewing Facility and the VM Dump Tool.

Table 4. DVF Commands to VM Dump Tool

DVF Commands	VM Dump Tool Function	Notes
ADDMAP		(1)
DUMPLOAD	DUMPLOAD	Exactly the same
	DUMPLD2	
DUMPSCAN	VMDUMPTL	
MAP		(1)
PRTDUMP	VMDUMPTL, DISPLAY	
TRACERED	TRACERED	Exactly the same
VIEWSYM	VMDUMPTL, SYMPTOM	

Table 5. DVF Macro Subcommands to VM Dump Tool

DVF Macro Subcommands	VM Dump Tool Function	Notes
DRESTORE		
DSAVE		
DVFSTACK	DVFSTACK	Use REXX Queue
FINDSTRG	FINDSTRG, LOCATE, LOCDISP, SETVAR	
INIT	DUMPNAME, INIT	
NOTE		Use REXX Queue
READSTRG	READSTRG, SETVAR	
SCAN		Use XEDIT SET PFx

Note that the VM Dump Tool functions DVFSTACK, FINDSTRG, INIT, and READSTRG should be used for migration purposes only.

Table 6. DVF Subcommands to VM Dump Tool

DVF Subcommands	VM Dump Tool Function	Notes
+ (plus symbol)	SCROLL	
- (minus symbol)	SCROLLU	
? (question mark)	F12	
= (equal symbol)	F12, Enter	
&name (ampersand)		
	ABSOLUTE (turn off PREFIXing)	
ACCLIST		(2)
	ANALYZE (display bit meanings)	

Table 6. DVF Subcommands to VM Dump Tool (continued)

DVF Subcommands	VM Dump Tool Function	Notes
AREGS	AREGS	
ASID		(2)
BACKWARD	SCROLLU	
BLOCK	BLOCK	
	CCWx (analyze CCW chain)	
CHAIN	CHAIN	
	CLOCKS (display clock values)	
	CODE (display abend code)	
	CONSOLES (display information about all user consoles)	
CMS	CMS	
CMSPOINT		(2)
CMSVIEW		(2)
CMSVIEW TRACE		(2)
	CP (issue CP command)	
CPEBK	CPEBK	
	CPEXITS (display information on CP Exits)	
	CPUUSE (display usage of CPUs)	
	CPVOLS (display status of all CP owned volumes)	
CPU	CPUID	
CREGS	CREGS	
	DESCRIBE (display meaning of an abend code)	
DISPLAY	DISPLAY	
DOSPOINT		(2)
DUMPID	DUMPTYPE	
	DUMPNAME (display the name of the dump)	
DUMPSCAN	VMDUMPTL	
END	FILE or QUIT	
FDISPLAY		(2)
FINDCPE	FINDCPE	
FINDMOD	MAP	
FINDUSER	FINDCPE CHAIN or CALLERS	
FORMAT		(2)
FORWARD	SCROLL	
FRAMETBL	FRAME	
	FRAMES (display statistics about frame usage)	
	FREGS (display floating registers)	
GDISPLAY		(2)
	GREGBASE (optional base for GREGS display)	
GREGS	GREGS	
HC	HEX	

Table 6. DVF Subcommands to VM Dump Tool (continued)

DVF Subcommands	VM Dump Tool Function	Notes
HELP	HELP	
	INDENT (indent output)	
	INDQ (indicate queues)	
	INSTR (disassemble instructions)	
	LINKS (display links for a user)	
HX	QUIT, QQUIT, FILE, FFILE	
INSPECT	SYMPTOM, GREGS, DISPLAY	(3)
IUCV	MAPIUCV	
	KEY (display storage key)	
LOCATE	LOCATE	
LOCATEUP		
	LOCDISP (display all occurrences of a string)	
	OFFSET (display from an address in previous data)	
OSPOINT		(2)
	PFXSAVE (format and display parts of PFX save areas)	
	PREFIX (set prefixing on for low storage)	
	PREG (display prefix register)	
	PSWS (format and display all PSWs)	
	QCPLEVEL (display information about system level, times)	
	QUERY (display settings of VM Dump Tool options)	
PRINT		(4)
QUIT	QUIT, QQUIT	
	RADIX (decode a Radix Tree)	
REAL	REAL	
REGS	REGS	
RIOBLOK	RDEVBK or RIO	
	RSCH (locate RDEV by subchannel)	
SCROLL	SCROLL	
SCROLLU	SCROLLU	
SELECT	included in TRACE	
	SETVAR (fetch storage from a macro)	
	SHRBKS (display SHRBKS for one or more users)	
SNAPLIST	SNAPLIST	
	SNTBKS (display system SNTBKs)	
	SPCON (display spooled console output)	
	SSASAVE (format and display SSA save areas)	
SYMPTOM	SYMPTOM	
TACTIVE		(2)
TIMEDIFF		
TLOADL		(2)

Table 6. DVF Subcommands to VM Dump Tool (continued)

DVF Subcommands	VM Dump Tool Function	Notes
TODCLOCK	TODCLOCK	
TRACE	TRACE	
TRACE (servers)		(2)
TRSAVE	TRSAVE	
TSAB		(2)
VIOBLOK	VDEVBK, VDEVS, or VIO	
VIRT	VIRTUAL	
VMDBK	VMDBK, VMDBKS	
	VMDSCAN (summary of all users in the system)	
VMLOADL		(2)
VPAIR		(2)
VREG		(2)
	VSCH (find RDEV by subchannel)	
VSTAT		(2)
XTRACE	TRACE	

Notes:

(1) The map is generated automatically by the VM Dump Tool. No explicit action by the user is required. The VM Dump Tool does not require, nor will it use, a CP nucleus map.

(2) Not applicable to a CP dump

(3) There is no direct equivalent to INSPECT. However most elements of the INSPECT function are provided separately. SYMPTOM, REGS, and TRACE provide the basic information from the dump. FINDCPE and BLOCK output can also be useful for certain types of problems.

(4) Use XEDIT PUT and then PRINT the file, or FILE to end the dump session to a dump log and print the DUMPLOG file.

Appendix D. Format of Block Data

BLOCK support consists of a number of elements to format and display the contents of control blocks and individual bytes, including:

- The BLOCK macro
- The ANALYZE subcommand
- A number of other subcommands and macros that also display bit definitions, including CALLERS, CCW, CPEBK, FRAME, KEY, RDEV, VDEV, and a number of individual trace entry formatters
- 32-bit base Block Definition Library, file HCQB4*xxx* VMDTDATA (where *xxx* is the version and release number of z/VM)
- 64-bit base Block Definition Library, file HCQB8*xxx* VMDTDATA (where *xxx* is the version and release number of z/VM)

  See the Release operand of the "VMDUMPTL Command" on page 24 for valid version and release numbers.
- Possible future service Block Definition Library file(s)
- Optional user-written Block Definition Library file(s)
- An optional user-modified Block-related profile, file VMDTBPRF XEDIT.
- A number of IBM-supplied REXX and XEDIT files which provide the underlying support needed.

Block Definition Library File Structure

A Block Definition Library File is a simulated Partitioned Data Set similar to a CMS MACLIB that contains a separate member for each control block and byte definition that can be formatted and displayed.

When a dump is entered, a list of block definition library files appropriate for that dump is determined from the characteristics of the dump, and may be modified by the VMDTSET BLKLIBS subcommand or a user-written exit. This list of files is checked during initialization to verify that each of these files is, in fact, available to the VM Dump Tool. If a file is not found, a warning message is issued and processing continues.

When a request for the display of a byte or control block is received, this list of library files is accessed, and each file in the list is searched for the byte or control block name requested. When the right name is found, the corresponding member of the Block Definition Library in which it was found is read from the file and used to control the subsequent display and formatting.

If desired, you can build a user library file to contain either modifications of IBM-supplied CP control blocks (not recommended, but is allowed) or locally-written control blocks. These should be placed in a separate Block Definition Library file that is searched ahead of the normal sequence.

The underlying structure of a Block Definition Library file is the same as a CMS MACLIB, but the file type is always VMDTDATA. A library file can be built or manipulated with the CMS MACLIB command or the MACLIB pipelines stage and then renamed to file type VMDTDATA for the VM Dump Tool to use.

Control Block Definition

A Block Definition Library file consists of one member per control block or byte. Each element of a member describes one field in the control block or one bit combination in the byte. All of this information is kept in a free-form format, then massaged into 80-byte records to fit in the VMDTDATA file.

Each control block definition consists of one entry per field in a variable length logical record:

- Displacement in the control block (decimal, origin 0)
- Duplication factor (decimal)
- Field length (decimal)
- Data type (one-character code)
- Field label (not limited in length, but it displays better if 14 characters or less)
- Comments (any length, all comments for one field on one line). When displayed, extra blanks will be removed and the data will be formed into paragraphs.

There is no specific maximum data length for a control block definition.

Bits/Codes Definition

A given byte contains either bits or a code. A designation of 'bits' means that each individual bit has its own meaning, independent of the meaning of other bits in the byte. The term 'code' is used when the combination of bits, as a unit, has a unique meaning.

The first line of the definition indicates the type and name of the byte that it describes. Use '*BITS' for a byte that contains bits, and '*CODES' for a byte that contains any of a series of codes. The second term on the first line is the name of the byte being described.

Subsequent lines for bits or codes definitions are all the same and have the following format:

- Columns 1 and 2 must contain the two-digit hexadecimal value that is being defined.
- The rest of the line contains the comments of what this bit or code value means. When displayed, extra blanks will be removed and the data will be formed into paragraphs.

The total of all characters in all lines of a byte or code definition is limited to a total of 4096 bytes in length. Data beyond that length will be truncated without notice. (See “Bits Definition File” on page 222 for a sample definition.)

Creating a Local Block Definition Library File

It is possible for a user to add a library of locally-written definitions or replacements for IBM-supplied definitions. The general steps to do this are as follows:

- Create a file describing the control block or byte
- 'Compress' this data so it will fit into a MACLIB member
- If you wish to use an existing Block Definition Library file, rename that file to *fn* MACLIB

- Use the MACLIB command to add the new definition to a Block Definition Library file
- Rename the resulting MACLIB back to *fn* VMDTDATA
- If you are creating a new Block Definition Library file, you must add logic to the Block-related user exit program, VMDTBPRF XEDIT, to include the name of the new file.

Each of these steps are reviewed in more detail in the following sections.

Create a Definition File

You may create the raw definition file with XEDIT or extract an existing definition and modify it.

If you use XEDIT, use the WIDTH parameter with a value that is large enough to hold the longest line you want to enter for one definition. If in doubt, make it bigger.

If you want to extract and modify an existing definition, see "Definition Extract Program" on page 222 for a sample.

Compress the Data to a *COPY File

The definition file must be massaged to combine all the input lines into one long line with a byte of hexadecimal 00 between logical lines. The result must be repackaged into 80-byte lines. Also, a *COPY line is needed at the beginning to tell MACLIB the name of the new member.

The following program fragment illustrates one way that this can be done:

```
fieldnm = 'sample'
fn = 'sample input a'
infile = 'MYDEF
Address COMMAND 'PIPE',
   '    <' infile,              /* read one definition */
   '|   strip',                 /* drop extra blanks */
   '|   join * x00',            /* one string per block */
   '|   deblock 80',            /* to right lrecl for maclib */
   '|   preface strliteral $*COPY' fieldnm'$', /* add *COPY line */
   '|   pad 80',                /* possibly pad out last record */
   '|   >' fn 'copy a fixed 80' /* write file for MACLIB cmd */
```

- If you wish to use an existing Block Definition Library file, rename that file to *fn* MACLIB.
- Use the MACLIB command to add the new definition to a Block Definition Library file
- Rename the resulting MACLIB back to *fn* VMDTDATA
- If you are creating a new Block Definition Library file, you must add logic to the Block-related user exit program, VMDTBPRF XEDIT, to include the name of the new file.

Samples/Examples

This section shows you samples and examples of some control block definitions.

Control Block Definition

The following is a sample definition (for the LCKBK control block). Note that the lines ending in ... have been shortened to fit this page. In the real definition, they extend farther to right to include the whole definition of each field.

```
  0 1 4 X LCKNEXT POINTER TO THE NEXT LOCK BLOCK
  4 1 4 X LCKQUE POINTER TO CPEXBLOK QUEUE
  8 0 8 D LCKNAME LOCK SYMBOL
  8 1 4 X LCKNAM1 FIRST FOUR BYTES OF SYMBOL
 12 1 4 X LCKNAM2 SECOND FOUR BYTES OF SYMBOL
 16 1 4 A LCKIDR11 R11 of lock requestor
 20 1 4 A LCKIDR14 R14 of lock requestor
 24 1 8 D LCKQQTOD TOD saved at last enqueue/dequeue. (Only stamped ...
 32 1 4 X LCKDQLEN Current length of queue waiting for this symbolic...
 36 1 4 X LCKDYTOD TOD saved at start of last "delay" period. It's t...
```

Bits Definition File

The following is an example of the definition of a byte used for bit definitions (for the CCTSTAT byte in the CCTBK control block). Note that these comments could be longer if need be.

```
*BITS CCTSTAT
80 RETRIEVE BUFFER IN PROGRESS
40 ABLE TO PROCESS SYNC SENDS
20 CP server interrupt in progress
10 IUCV INTERRUPT IS STACKED
```

Codes Definition File

The following is an example of the definition of a byte used for code definitions (for the LNKSTAT byte in the LNKBK control block). Note that these comments could be longer if need be.

```
*CODES LNKSTAT
01 - Idle
02 - Busy
03 - Write in progress
04 - Read in progress
05 - Reset in progress
06 - Handling link attention
07 - Fatal I/O on link
08 - Initialization in progress
08 - Deactivation in progress
```

Definition Extract Program

If you wish to build a definition that is similar to an existing one, the following REXX program fragment can be used to extract an existing definition from a Block Definition Library file and put into the appropriate source format.

```
/*----------------------------------------------------------------*/
/* Program fragment to extract a single definition                */
/*----------------------------------------------------------------*/
libfn = 'hcqb4250'
libft = 'vmdtdata'
element = 'cctbk'
outfile = 'sample output a'

Address COMMAND 'PIPE (endchar ?)',
   '|   cms type' libfn libft '* (member' element,
   '|   locate 1',                 /* drop blank lines */
   '|   drop first',               /* drop *COPY file line */
   '|   drop last',                /* drop /0000/ line */
```

```
'|   join *',                  /* to one big line */
'|   split at string x00',     /* split at line boundaries */
'|   >' outfile                /* to an output file */
```

Appendix D

Appendix E. Macro Message Identifiers

Only 3 characters are provided in which to identify the module or macro that issues a message. Because many macro names are longer that 3 characters, messages issued by VM Dump Tool macros use a coded 3-character identifier. The following table provides the 3-character module identifier in a message that relates to the macro that actually issued that message.

Table 7. Three-Character Module Identifier for Messages

3-Character Module ID	Actual Macro Name
XBF	HCQBLKFM REXX
XBK	BLOCK VMDT
XBR	HCQBLKRD REXX
XBV	HCQBLKRV REXX
XCF	HCQBLKCF REXX
XCL	CALLERS VMDT
XCN	CONSOLES VMDT
XCU	CPUUSE VMDT
XCV	CPVOLS VMDT
XDS	DESCRIBE VMDT
XEX	CPEXITS VMDT
XFC	FINDCPE VMDT
XFE	HCQREAD XEDIT
XFR	FRT2MAIN VMDT
XGP	HCQGPSW VMDTTRC
XIB	HCQINBIT VMDT
XIM	HCQINMAP VMDT
XLD	LOCDISP VMDT
XLN	LINKS VMDT
XLT	LASTTRAN VMDT
XMI	MAPIUCV VMDT
XPR	HCQXPROF XEDIT
XPS	PFXSAVE VMDT
XQC	QCPLEVEL VMDT
XRA	REAL VMDT
XRB	HCQRLB VMDT
XRC	HCQRLC VMDT
XRD	RADIX VMDT
XSB	SNTBKS VMDT
XSC	SPCON VMDT
XSH	SHRBKS VMDT
XSI	HCQSCSI VMDTTRC

Table 7. Three-Character Module Identifier for Messages (continued)

3-Character Module ID	Actual Macro Name
XSL	SNAPLIST VMDT
XSS	SSASAVE VMDT
XSY	SYMPTOM VMDT
XTI	HCQTIMOD VMDTTRC
XVD	VDEVS VMDT
XVM	VMDBKS VMDT
XVR	VIRTUAL VMDT
XVS	VMDSCAN VMDT
X02	HCQT0200 VMDTTRC
X03	HCQT0300 VMDTTRC
X33	HCQT3300 VMDTTRC
X83	HCQT8300 VMDTTRC

Appendix F. Sample Programs Associated with the VM Dump Tool

The following macros are provided as sample programs only. The basic functions should work; however, they have not been thoroughly tested.

PKINIT Macro

Purpose

The PKINIT macro displays register and PSW information from a Performance Toolkit dump.

Examples

Typical use of and output from the PKINIT macro.

```
>>> pkinit
----- Registers at abend time -----

   General Purpose Registers
   R0 0000001C                  R8 00D332B0 FCXBAS+2000
   R1 00D34D52 FCXBAS+3AA2      R9 00D342B0 FCXBAS+3000
   R2 01F970C4                  RA 03E90000
   R3 000078C8                  RB 00D312B0 FCXBAS
   R4 80F524C6                  RC 00D322B0 FCXBAS+1000
   R5 03E92000                  RD 00D33100 FCXBAS+1E50
   R6 00000001                  RE 80D34646 FCXBAS+3396
   R7 00001378                  RF 00D32F30 FCXBASD

   Control Registers
   CR  0 000010E0 00000000 00000000 00000000
   CR  4 00000000 00000000 FF000000 00000000
   CR  8 00000000 00000000 00000000 00000000
   CR 12 00000000 00000000 DF000000 00000000

   Access Registers
   AR  0 00000000 00000000 00000000 00000000
   AR  4 00000000 00000000 00000000 00000000
   AR  8 00000000 00000000 00000000 00000000
   AR 12 00000000 00000000 00000000 00000000

   Floating Point Registers
   FR  0 4100000000000000
   FR  2 0000000000000000
   FR  4 0000000000000000
   FR  6 0000000000000000

   PFX VALUE 00000000

   PSWs
   EXT 4000 18 Old 00080000 80D33648 FCXBAS+2398
            58 New 00080000 80F22FD8
   SVC 00C9 20 Old 000C0000 00F12838
            60 New 000C0000 80F116D6
   PRG 0001 28 Old 00E80000 80E89C66 FCXWWO+B76
                   PgmIntCode = Operation exception
            68 New 00080000 81381938
   MCH      30 Old 00000000 00000000
                   MCIC = 00000000 00000000
            70 New 00080000 80F3E9E8
   I/O      38 Old 03EC0000 80D31E60 FCXBAS+BB0
            78 New 00080000 80F10DD8
```

```
----- Registers stored by CMS at Program Check Time -----

   pgmsect at 0654
   psave at 00007944

   General registers from 00007944
   R0 01F75355                   R8 01F971FE
   R1 00000000                   R9 00000000
   R2 01F970C4                   RA 03E90000
   R3 00000000                   RB 00E890F0 FCXWWO
   R4 01F75367                   RC 00E8A0F0 FCXWWO+1000
   R5 01F73F78                   RD 00E8A598 FCXWWO+14A8
   R6 01F96FC8                   RE 00E8A696 FCXWWO+15A6
   R7 00000000                   RF 00000000
```

PKSV Macro

Purpose

The PKSV macro displays a standard OS-format save area from a Performance Toolkit dump.

Operands

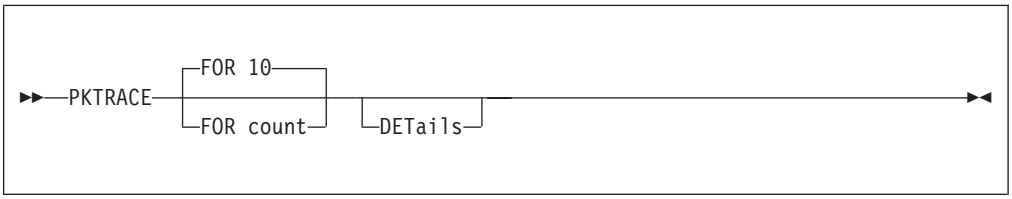
address
is the address of the save area to be displayed.

Examples

Typical use of and output from the PKSV macro.

```
>>> pksv 00D33100
R0 00000000                  R8 00D332B0 FCXBAS+2000
R1 00EE8EA0                  R9 00D342B0 FCXBAS+3000
R2 01F970C4                  RA 03E90000
R3 00008070                  RB 00D312B0 FCXBAS
R4 80F524C6                  RC 00D322B0 FCXBAS+1000
R5 01F73F78                  RD 00000000
R6 00000001                  RE 80D32F26 FCXBAS+1C76
R7 00001378                  RF 00D32F30 FCXBASD
```

PKTRACE Macro



Purpose

The PKTRACE macro is used to display data from the trace table in a Performance Toolkit dump.

Operands

FOR count
specifies how many trace entries to display.

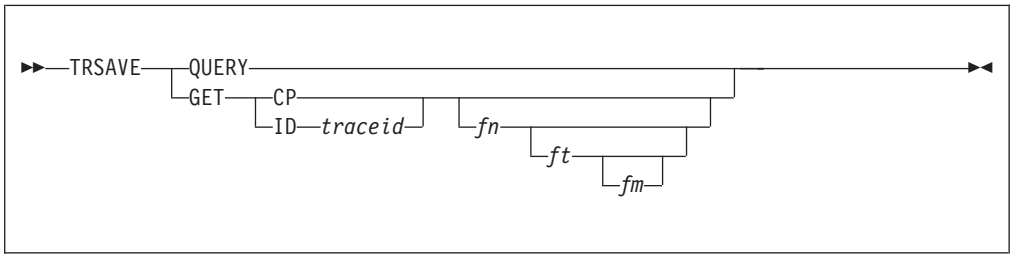
DETails
specifies that details of the trace table should be displayed.

Examples

Typical use of and output from the PKTRACE macro.

```
>>> pktrace
03F3AC18 9D62 Call fr FCXBAS+1C76 to FCXBAS+1CBA fcn DUMP
03F3AC08 9D62 Call fr FCXWWW+2338 to FCXWWO+8A6 fcn WWOH
03F3ABF8 9D62 Rtrn to FCXWWW+2442 fr FCXMSG+716 fcn MSG
03F3ABE8 9D62 Rtrn to FCXMSG+15BE fr FCXMSG+AD2 fcn MSGS
03F3ABD8 9D62 Call fr FCXMSG+15BE to FCXMSG+9D2 fcn MSGS
03F3ABC8 9D62 Call fr FCXWWW+2442 to FCXMSG+3A fcn MSG
03F3ABB8 9D62 Rtrn to FCXWWW+2442 fr FCXMSG+716 fcn MSG
03F3ABA8 9D62 Rtrn to FCXMSG+15BE fr FCXMSG+AD2 fcn MSGS
03F3AB98 9D62 Call fr FCXMSG+15BE to FCXMSG+9D2 fcn MSGS
03F3AB88 9D62 Call fr FCXWWW+2442 to FCXMSG+3A fcn MSG

>>> pktrace for 4
03F3AC18 9D62 Call fr FCXBAS+1C76 to FCXBAS+1CBA fcn DUMP
03F3AC08 9D62 Call fr FCXWWW+2338 to FCXWWO+8A6 fcn WWOH
03F3ABF8 9D62 Rtrn to FCXWWW+2442 fr FCXMSG+716 fcn MSG
03F3ABE8 9D62 Rtrn to FCXMSG+15BE fr FCXMSG+AD2 fcn MSGS

>>> pktrace for 5 det
baseaddr     : 00007944
segment reg  : 10
segreg@      : 0000796C
segment addr : 03E90000
Trace start  : 03F39FF8
Trace end    : 03F3AFF8
Trace current: 03F3AC28
03F3AC18 9D62 Call fr FCXBAS+1C76 to FCXBAS+1CBA fcn DUMP
03F3AC08 9D62 Call fr FCXWWW+2338 to FCXWWO+8A6 fcn WWOH
03F3ABF8 9D62 Rtrn to FCXWWW+2442 fr FCXMSG+716 fcn MSG
03F3ABE8 9D62 Rtrn to FCXMSG+15BE fr FCXMSG+AD2 fcn MSGS
03F3ABD8 9D62 Call fr FCXMSG+15BE to FCXMSG+9D2 fcn MSGS
```

TRSAVE Macro



Purpose

Use the TRSAVE macro to display information regarding enabled trace IDs and active CP trace or to extract the trace data from an enabled trace ID or active CP trace. The extracted trace data is put into a CMS file for later input to TRACERED.

Operands

QUERY
specifies that information regarding all enabled trace IDs and active CP trace should be displayed.

GET
specifies that the trace data is to be extracted.

CP specifies that trace data is to be extracted from the active CP trace.

ID *traceid*
specifes that trace data is to be extracted from the enabled *traceid*.

fn specifies the file name of the CMS file to be created with the extracted trace data. The default file name is the file name of the dump.

ft specifies the file type of the CMS file to be created with the extracted trace data. The default file type is the trace ID, or CP for CP trace.

fm specifies the file mode of the CMS file to be created with the extracted trace data. The default file mode is A.

Usage Notes

1. This macro requires enough R/W disk space to contain the output file.
2. A file of fixed length records is created when TRSAVE extracts CP trace data. A file of variable length records is created when TRSAVE extracts TRSOURCE trace data. These files need to remain in their original fixed or variable length format for proper input to TRACERED.
3. The data generated can be submitted to TRACERED via the unsupported input operand FILE:

   ```
   TRACERED ... FILE fn ft fm
   ```

   See the *z/VM: CP Commands and Utilities Reference* for more information about TRACERED.

Other Samples and Examples for VM Dump Tool

The following is a list of other sample/example VM Dump Tool macros that are included with the VM Dump Tool and a brief description of each. These are all in HELP component VMDT. See the corresponding HELP file for more details about what they do, the syntax required to use them and sample output.

Macro	Description
ACTIVEIO	Displays summary of active real & virtual I/O
ACTRIO	Displays summary of active real I/O
ACTVIO	Displays summary of active virtual I/O
ASCBKS	Displays all ASCBKs in the system
ASSERT	Decodes clues in an ASR001 abend
BBLOCK	Boils BLOCK output to reduce big data areas
BOXED	Displays list of devices in 'boxed' status
CALLSEQ	Displays calling sequence from s/a address
CCPV	Displays dasd info, related block addrs
CKPTIL	Displays PTIL Lock status for all VMDBKs
CPECHNS	Displays outstanding CPEBKs, calling sequences
CPE2FILE	Writes list of CPEBK chains to a file
CPVOLSB	Displays CPVOLs & associated device information
DGA	Displays guest virtual storage
DSCANSUM	Displays demand scan information
DUMPSUM	Displays dump info summary, CPUs Stgsize etc
DX	Display eXtended, format data per usage
EDEVS	Displays summary of SCSI-related devices
ELIST	Displays all VMDBKs on the eligible list
ENGINES	Displays types of engines present in the dump
ENV	Displays the operating environment
EPS	Displays all entry points for a given module
EXPBKCTS	Captures some counts from EXPBKs
FMTASCE	Displays the content & meaning of an ASCE
FRMAVL	Displays information about available frames
GETASCES	Displays summary of ASCE for each VMDBK
GUESTLAN	Displays summary of LANBKs
HANGSUM	Displays summary of system hang
HTT001	Deciphers an HTT001, HTT002 or SIT002 abend
IOLHANG	Finds threads hung in IOL, interprets details
IOVVDEV	Associates IOV threads to virt device/user
LASTCPE	Displays summary of each outstanding thread
LCPUA	Displays logical CPU addr for each CPU
LOCKAT	Interprets a logical defer lock
LOCKQ	Displays work queued on a lock
LOGREGS	Displays regs and PSW from CPU logout area
MCIC	Decodes MCIC (Mach Ck Interrupt Code)
MODLVL	Displays the level of a module from the dump
PAV	Displays list of all PAV devices
PAVSUM	Displays summary of all PAV devices
PFXALL	Displays the same field from all PFXPGs
PLDVS	Displays PLDV contents for master, each CPU
PSWALL	Displays same PSW from all CPUs
PSWFMT	Interprets and formats the first word of a PSW
PTE	Displays status about a page table entry
PUSHTHRU	Displays push through stacks & queues
READYSUM	Displays info about users in ready status

Macro	Description
SCHEDLK	Displays status of the scheduler lock
SECUSERS	Displays all users with secondary users defined
SHARES	Displays some share-related fields from VMDBKs
SRMBK	Displays system resource info from the SRMBK
STGSIZE	Displays storage size
SUBBKS	Displays summary of sub-pool storage usage
SUBX	Displays summary of usage for 1 SUBBK
SUPERCPE	Displays CPE chains for all VMDBKs for a user
SXSCPPGS	Displays summary of in-use SXS CP Pages
SXSTABLE	FRAMES subcommand for the SXSTE table
SYN002	Displays info about a SYN002 abend
SYSSVC	Displays info about IUCV System Services
SZREQ	Displays SZREQs for an EDEV
TIMEOUTS	Displays all TRQBKs for guest timeouts
TRALL	Displays the last 5 trace entries from each CPU
TRBYTIME	Displays last trace entry for all CPUs by time
TRCBKS	Displays summary of all TRCBKs in the dump
TRCIOSUM	Displays summary of I/O in the trace table
TRCPAGES	Displays trace page chain for each CPU
TRQBKS	Displays summary of all active TRQBKs
TSKBK	Displays current TSKBK, regs & PSWs
UFODIST	Displays dist of frames < & >2G for a user
USERLANS	Displays summary of LAN connections
USERSUM	Displays summary of user status
USPACE	Set up to display a user address space
VMDALL	Displays same field for all VMDBKs
VMDBKSUM	Displays status of one VMDBK
VMDPLDV	Displays users who are in a PLDV
VMDVTIME	Displays VMDVTIME field for each user

The following is a list of other sample/example VM Dump Tool pipes stages that are included with the VM Dump Tool and a brief description of each. See the corresponding HELP file for more details about what they do, the syntax required to use them, input data stream requirements and sample output.

Macro	Description
AASCBKS	Returns list of ASCBKs for a VMDBK
ACPEBKS	Returns list of all CPEBKs
ACPECHNS	Returns list of all CPEBK chains
ACPVOLS	Returns list of all CPVOLs
ADR2MOD	Returns module name for an address
AIUCVBKS	Returns list of all major IUCV blocks
ALANBKS	Returns list of all LANBKs
AMODULES	Returns list of modules in the map file
ANALYZE	Displays bit meanings for a given byte & value
APLDVS	Returns list of all CPUs and PLDVs
ARDEVBKS	Returns list of all RDEVs
ATRCBKS	Returns list of all TRCBKs
ATRQBKS	Returns list of all TRQBKs
AVDEVBKS	Returns list of VDEVs for a given VMDBK
AVMDBKS	Returns list of all primary VMDBKs
BLANKEM	Blank given columns of duplicate data
BOILBLK	Extracts mostly comments from BLOCK output
CHAIN	Follows chain via CHAIN subcommand

Macro	Description
CHAIN8	Follows chain via CHAIN8 subcommand
CPEMSTR	Returns flag of whether CPEBK requires master
CPESEQ	Returns calling sequence for a CPEBK
CPUS	Returns list of all CPUs, PFXPGs
CPUTYPE	Returns CPU type from PFXPG@
CPVOLSC	Display information from the CPVOL
CTONES	Counts one bits in a string of hex data
CTSAME	Counts duplicate values in a field
CVTX2D	Converts hex data to decimal
FRMTES	Returns FRMTE content from the Frame Table
GETBYTE	Returns a byte from a given addr & displ
GETDATA	Returns a string from a given addr & displ
GETDWORD	Returns a d-word from a given addr & displ
GETHALF	Returns a halfword from a given addr & displ
GETSVREG	Returns a reg value from a save area
GETWORD	Returns a word from a given addr & displ
HCQGFLD	Returns field displacement from BLOCK data
HEXLIST	Returns a range of hex values per incr & count
HEXSORT	Sorts data in hex
INAPLDV	Returns flag of whether VMDBK is in a PLDV/not
IUCVQ	Returns IUCV message queue for a VMDBK
PFXTYPE	Returns CPU type tag for given PFXPG
PLDVQ	Returns PLDV queue from PLDV address
PLDV2CPU	Returns CPU number from PLDV address
RADIXX	Expands a radix tree
RDEVCLAS	Returns RDEVCLASS for given RDEV addr
SXSTES	Returns SXSTE content from the SXSTE table
SYSCM	Returns address of SYSCM or contents of a field
TOD2TIME	Returns date/time for a TOD value
TWOGFRM	Returns < >2g flag from FRMTE addresss
USER2VMD	Returns primary VMDBK addr for a userid
VMDLCYCL	Returns chain of local VMDBK cyclic list
VMDSLIST	Returns status of a VMDBK
VMD2USER	Returns userid for a VMDBK address
V2CPVOL	Returns CPVOL addr from volume index
XADD	Returns sum of input word and fixed value
XAND	Returns the AND of a value with a constant

Appendix G. Using the Trace Format Definition Table

The Trace Format Definition Table is made up of one or more Trace Definition Files which contain specifications for how trace entries should be formatted. The name of the default Trace Definition File can be displayed with the VMDTQRY TRACE LIBRARY subcommand. The name of the default library follows the string 'default>', as in the following example:

```
>>> dtq trace lib
    LIB  set to Default> HCQTR540
```

The name of the file must have the form 'fn VMDTDATA' and the content must conform to the following sections. You can tell the VM Dump Tool to use a local Trace Definition File with:

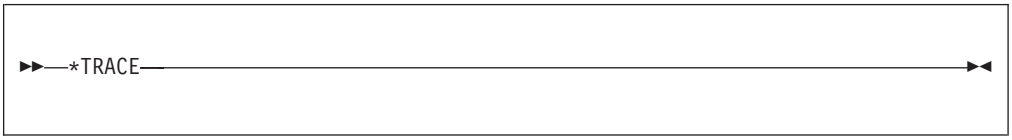
```
vmDTSet TRAce LIBrary fn <fn ...>
```

Summary of Trace Definition File Contents

Statement Type	Function Type
*TRACE	This specification identifies this file as a Trace Definition File.
TERM	This specifies hex values and the strings associated with them to be displayed for particular terms.
Trace Entry Definition	This specifies hex values and the strings associated with them to be displayed for particular terms.

*TRACE Statement

A *TRACE statement specifies that this is a Trace Definition File.



Note:

1. A *TRACE statement must be the first line in a Trace Definition File.
2. The asterisk must be in column 1.
3. There must be no blank between the asterisk and 'trace'.
4. The string 'trace' may be in any combination of upper and lower case.

TERM Definition

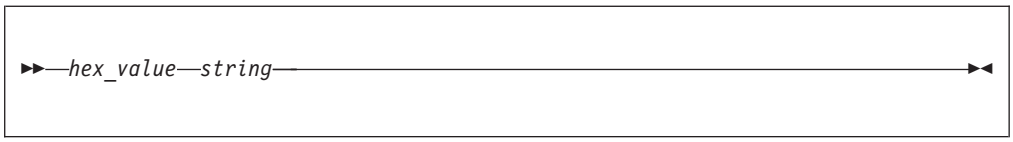
Syntax, first line.

Where:

TERM
specifies that this is a Term Definition Statement. TERM must start in column 1.

termname
is the one- to eight-character name of a term referenced in a Trace Entry Definition (see below)

Syntax, second and subsequent lines.



Where:

hex_value
specifies a two- or four-character hexadecimal value which can be associated with this term. This must start in column 2 or beyond.

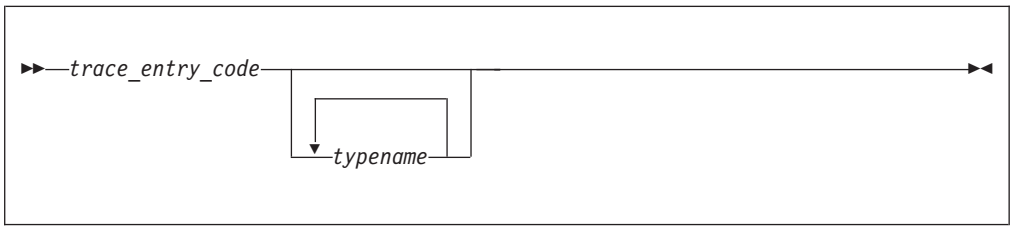
string
specifies a one- to thirty-two-character string which specifies the meaning of this hex value. This string must be enclosed in single quotation marks (').

Note:

1. A term definition is terminated by another term definition, a trace entry definition, or the end of the file.
2. A term definition does not have to precede references to it by trace entry definitions.
3. If the same term is defined again, the second and subsequent definitions are ignored.

Trace Entry Definition

Syntax, first line.

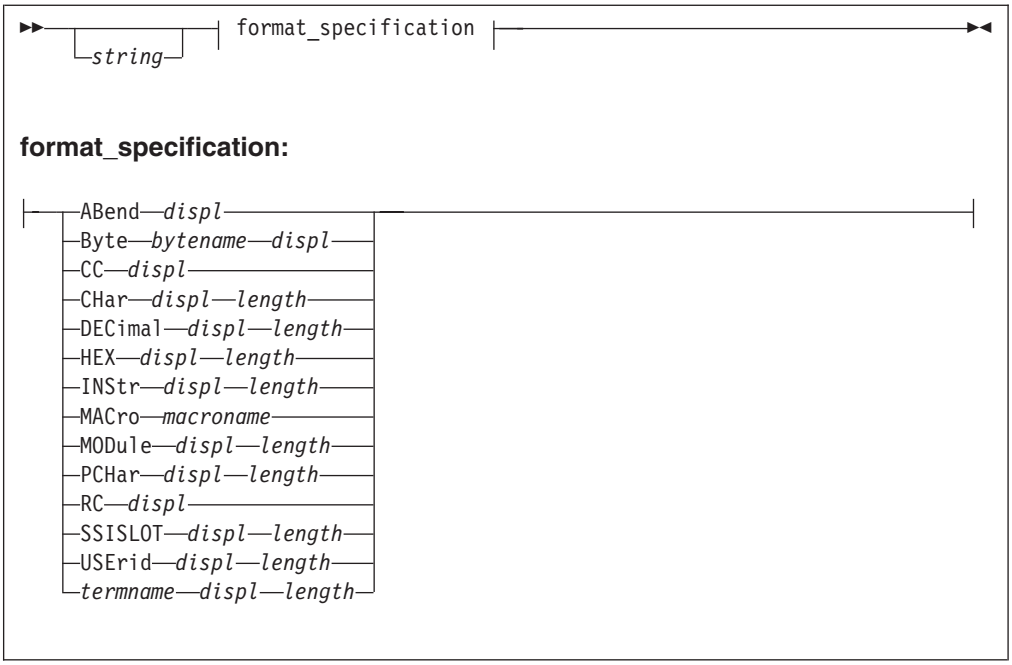


Where:

trace_entry_code
specifies the four-digit hexadecimal trace entry code for which the format is being defined.

typename
specifies one or more one- to eight-character type names with which this trace entry is associated.

Syntax, second and subsequent line.



Note:

1. If the same trace entry code has been defined previously, this one is ignored without an error message.
2. The trace entry code must be exactly four hexadecimal digits in length and must start in column 1.

Format Specifications

A format specification is free form and can start in any column except column 1. Each line has the following format:

string
: specifies an optional string to be displayed. It must be enclosed in single quotation marks. The maximum string length allowed is 32.

displ
: specifies the absolute displacement in the trace entry where the data to be formatted is located.

  Allowed values for a type-74 trace entry include:

```
any hex value between 0 and 1F.
TTEDATA0 - data word 0, displ 0C
TTEDATA1 - data word 1, displ 10
TTEDATA2 - data word 2, displ 14
TTEDATA3 - data word 3, displ 18
TTEDATA4 - data word 4, displ 1C
```

  In addition, a byte number B0-B3 can be included after any displacement keyword above:

```
TTEDATA0B0 - data word 0, byte 0, displ 0C
TTEDATA0B1 - data word 0, byte 1, displ 0D
TTEDATA0B2 - data word 0, byte 2, displ 0E
TTEDATA0B3 - data word 0, byte 3, displ 0F
```

  Allowed values for a type-75 trace entry include:

```
any hex value between 0 and 3F.
TTEGDAT0 - data word 0, displ 10
TTEGDAT1 - data word 1, displ 18
TTEGDAT2 - data word 2, displ 20
TTEGDAT3 - data word 3, displ 28
TTEGDAT4 - data word 4, displ 30
TTEGDAT5 - data word 5, displ 38
```

In addition, a byte number B0-B7 can be included after any displacement keyword above:

```
TTEGDAT0B0 - data word 0, byte 0, displ 10
TTEGDAT0B1 - data word 0, byte 1, displ 11
TTEGDAT0B2 - data word 0, byte 2, displ 12
TTEGDAT0B3 - data word 0, byte 3, displ 13
TTEGDAT0B4 - data word 0, byte 4, displ 14
TTEGDAT0B5 - data word 0, byte 5, displ 15
TTEGDAT0B6 - data word 0, byte 6, displ 16
TTEGDAT0B7 - data word 0, byte 7, displ 17
```

length
specifies the number of bytes that are to be processed for this field.

ABEnd *displ*
specifies that four bytes at the indicated displacement should be displayed as an abend code (MODnnn where MOD is the module name and nnn is the specific abend code.

A *displ* value is required.

Example:

The trace entry in hex:

```
_7C011340 +0000 7400F848 7275AC40 00320200 D7E9C501 *..8.... ....PZE.
_7C011350 +0010 40E2E5C3 00020000 04040000 00351612 * SVC.............
```

The trace definition table entry for this data:

```
abend     ttedata0
```

The abend data is taken from displacement ttedata0 (X'0C'), which is D7E9C501. The abend formatter reformatted this to PZE001.

The output of the trace command for this entry and this data:

```
7C011340 05:42:08 PZE001 ...
```

BYTe *bytename displ*
specifies that the bit meanings for one byte of data at the indicated displacement should be displayed. The format of the output is the same as from the ANALYZE subcommand.

bytename
is the name of a field in a control block which contains a single byte of flags or a code. Example: VMDRSTAT, which is a byte field in the VMDBK control block.

The *bytename* and *displ* values are both required.

Example:

Trace entry data:

```
_7C0111C0 +0000 7580BE35 F8487274 B9000000 00320501 *....8...........
_7C0111D0 +0010 C0861797 92C0C0C0 00000000 0101DDD8 *{f.pk{{{.......Q
_7C0111E0 +0020 00000000 056FCA98 00000000 00000000 *.....?.q........
_7C0111F0 +0030 00000000 00000000 00000000 00000000 *................
```

The trace definition table entry for this data:

```
byte           rdevdp    ttegdat0b4
```

The bits for the byte at displacement ttegdat0b4 (X'14'), X'92' are displayed.

TRACE command output:

```
7C0111C0 05:42:08 ...
     Bits  defined in RDEVDP           (92)
       80 PATHS ARE CURRENTLY GROUPED
       10 USE ALTERNATE PATH GROUP ID
       02 MULTIPATH MODE DP WAS ESTABLISHED
```

CC *displ*
specifies that bits 2 and 3 of the byte at the indicated displacement should be displayed as a condition code.

The *displ* value is required.

Example:

Trace entry data:

```
_7C071CC0 +0000 7400F848 67CE8140 00322C00 00000000 *..8...a ........
_7C071CD0 +0010 10D7E7D8 06286200 802267D4 8035103C *.PXQ.......M....
```

The trace definition table entry for this data:

```
cc        ttedata1    1
```

Bits 2-3 at displacement ttedata1 (X'10') is formatted to a condition code. The data at X'10' is 10, which is 0001 0000. Bits 2-3 are 01.

TRACE command output:

```
7C071CC0 05:42:08 ... cc1
```

CHAr *displ length*
specifies that the data at the indicated displacement and length should be displayed as character data.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7E4A2660 +0000 74009B9B 32F096A0 00344010 F0F0E7C1 *.....0o... .00XA
_7E4A2670 +0010 8010A840 041C9000 00A4A000 7E58A000 *..y .....u..=...
```

The trace definition table entry for this data:

```
'mode'    char    ttedata0b2    2
```

The string 'mode' is displayed. Then the data at displacement ttedata0b2 (X'0E') for two bytes is displayed as character data.

TRACE command output:

```
7E4A2660 CPU 0002 ... mode XA
```

DECimal *displ length*
specifies that the data at the indicated displacement and length should be displayed as decimal data.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7FB85CA0 +0000 74009C11 8559A9C0 00300600 4C4C4C4C *....e.z{....<<<<
_7FB85CB0 +0010 0000000A 030786F0 04378000 80396252 *......f0........
```

The trace definition table entry for this data:

```
'Obtain'       decimal   ttedata1    4
```

The four bytes of data are obtained from displacement ttedata1 (X'10'), X'0000000A' are formatted as a decimal number. Leading zeros are suppressed.

TRACE command output:

```
7FB85CA0 10:57:07 Obtain 10 ...
```

HEX *displ length*
specifies that the data at the indicated displacement and length should be displayed as hexadecimal data. If the length is exactly eight bytes, then an underscore will be added to the data between the first and second words.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7FA00060 +0000 740094C6 F0C9A84C 00000C02 030E1600 *..mF0Iy<........
_7FA00070 +0010 80C476CE 08E6B000 80000000 00000000 *.D...W.........
```

The trace definition table entry for this data:

```
'opsw'         hex       ttedata0    8
```

The eight bytes from displacement ttedata0 (X'0C') are formatted and displayed in hex.

TRACE command output:

```
7FA00060 14:18:08 ... opsw 030E1600_80C476CE
```

INStr *displ length*
specifies that the data at the indicated displacement and length should be disassembled into an instruction.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7FD89B40 +0000 7580BDCF 5F37EA49 72200010 0000B504 *....¬...........
_7FD89B50 +0010 00000000 00010000 00000000 06464000 *.............. .
_7FD89B60 +0020 00081000 80F10EA8 00000000 00000000 *.....1.y........
_7FD89B70 +0030 8280B235 D0900000 00000000 00000000 *b...}...........
```

The trace definition table entry for this data:

```
'instr'         inst      ttegdat4b2  6
```

The data at displacement ttegdat4b2 (X'32') for a length of 6 bytes, B235D0900000, is fetched and formatted as an instruction.

TRACE command output:

```
7FD89B40 CPU 0009 ... instr TSCH  X'90'(R13)
```

MACro *macroname*
specifies the name of a macro that should be invoked to format part or all of a trace entry.

The *macroname* value is required.

Example:

Trace entry data:

```
_7D494620 +0000 74005F37 EA49CBC9 00003310 E4C34040 *..¬....I....UC
_7D494630 +0010 00000000 05375000 05CCB800 001C3900 *......&.........
```

The trace definition table entry for this data:

```
macro     hcqt3310
```

In this example, the trace entry formatter invokes the macro named 'HCQT3310 VMDTTRC'. The macro interprets some of the data at X'10' in the trace entry, determines what data should be displayed, and then displays the CC/RC or 'To' address.

TRACE command output:

```
7D494620 22:14:14 ... to HCPIUF0
```

See "Writing Trace Macros" on page 248, for more information.

MODule *displ length*
specifies that the data at the indicated displacement and length should be used as an address and resolved to a module name, entry point name, or module name and displacement if possible. If the address can not be resolved, it is displayed as a hexadecimal value.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7D4642C0 +0000 74005F37 EAF2FD8E 00000600 4C4C4C4C *..¬..2......<<<<
_7D4642D0 +0010 00000005 0537DFC8 00002000 8014244C *.......H.......<
```

The trace definition table entry for this data:

```
'by'          module    ttedata4    4
```

The four bytes of data at displacement ttedata4 (X'1C'), 8014244C, are used as an address and resolved like the MAP subcommand to HCPERP+131C.

TRACE command output:

```
7D4642C0 CPU 0004 ... by HCPERP+131C
```

PCHAr *displ length*
specifies that the data at the indicated displacement and length should be displayed as character data enclosed in parentheses.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7FD197C0 +0000 74005F37 EAF0DD54 00000700 4CC7E2C4 *..¬..0......<GSD
_7FD197D0 +0010 00000005 0537DF48 05375000 800CBAEE *.........&.....
```

The trace definition table entry for this data:

```
'dw'          pchr      ttedata0b1  3
```

The string 'dw' is displayed. Then the data at displacement ttedata0b1 (X'0F'), C7E2C4, is fetched and displayed enclosed in parentheses. If the string <<< is found, it is converted to ???.

TRACE command output:

```
7FD197C0 22:14:14 ... dw (GSD)
```

RC *displ*
specifies that the four-byte value at the indicated displacement should be displayed as a return code.

The *displ* value is required.

Example:

Trace entry data:

```
_7D4643C0 +0000 74005F37 EAF3130E 00002C00 00000008 *..¬..3.........
_7D4643D0 +0010 10D9C2D2 05CCA800 80142568 802ABACA *.RBK..y.........
```

The trace definition table entry for this data:

```
rc        ttedata0
```

The value at displacement ttedata0 (X'0C'), 00000008, is fetched from the dump. If the value is less than 4,096, it is displayed as RC=n. If the return is larger than 4095, then it is displayed as 'RC>fff'

TRACE command output:

```
7D4643C0 22:14:14 ... rc=8
```

SSISLOT *displ length*
specifies that a slot number from the indicated displacement and length should be used to look up an SSI member's name.

The *displ* and *length* are both required. Length may be 1 or 2.

Example:

Trace entry data:

```
_03E102C0 +0000 7580C671 E42400FD 15820000 03631951 *..F.U....b......*
_03E102D0 +0010 00000000 005A0F44 D6D7C5D9 C1E3D5E2 *.....!..OPERATNS*
_03E102E0 +0020 00030180 01800000 00000000 00000000 *................*
_03E102F0 +0030 00000000 00000000 00000000 00000000 *................*
```

The trace definition table entry for this data:

```
'from'         ssislot   ttegdat2b0  2
```

The string 'from' is displayed. Then the two bytes starting at displacement ttegdat2b0 (X'0003') are used as an index into the list of SSI members and the member name is displayed as character data.

TRACE command output (assuming the member in slot 3 is named PLEXSYS3):

```
from PLEXSYS3
```

USErid *displ length*
specifies that a VMDBK address from the indicated displacement and length should be used to find and display the corresponding user ID from storage.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7D494700 +0000 74005F37 EA49F009 00000600 4C4C4C4C *..¬...0.....<<<<
_7D494710 +0010 00000010 0405F4D8 05375000 80222E9C *......4Q..&.....
```

The trace definition table entry for this data:

```
user      ttedata3    4
```

The VMDBK address, 05375000, is fetched from displacement ttedata3 (X'18'), the userid from the VMDUSER field of that VMDBK is fetched and displayed.

TRACE command output:

```
7D494700 22:14:14 ... OPERATOR
```

Note:

1. By convention, the default Trace Definition Table which is shipped with the VM Dump Tool (HCQTRACE VMDTDATA) always displays the VMDBK address and then the userid.

2. If the user ID has been listed in either the FOCUS or USER operand of the current TRACE subcommand, and there are multiple VMDBKs for this user, then a suffix will be added to the user ID which is displayed. (See the 'FOCUS' operand of the TRACE subcommand for more information).
3. The FOCUS and USER operands of the TRACE command depend on the USERID value stored in the trace definition table, NOT the VMDBKs. The USERID parm should be used in the trace definition file for every VMDBK address in a trace entry, in addition to the VMDBK parm.

termname displ length
specifies the name of a 'term'. The hexadecimal value of the data from the trace entry will be displayed.

The *displ* and *length* values are both required.

Example:

Trace entry data:

```
_7D7503A0 +0000 74005F2A 7FD88086 00001430 076E3B38 *..¬."Q.f.....>..
_7D7503B0 +0010 00078700 076E3AD8 05FB3948 076D7000 *..g..>.Q....._..
```

The trace definition table entry for this data:

```
'cctbk'        cctbk     ttedata2    4
```

The hexadecimal value at displacement ttedata2 (X'14'), 076E3AD8, is fetched and displayed.

TRACE command output:

```
7D7503A0 CPU 0000 ... cctbk 076E3AD8
```

TRACE command output with search for CCTBK:

```
>>> trace for 1 cctbk 076E3AD8 merge
   7D7503A0 CPU 0000 APPC/VM Int vmdbk 076D7000 PERFSVM
                     iucv 076E3B38 cctbk 076E3AD8 iparml 05FB3948
                     path 0007 int-type 87 flag1 00
```

Note:

1. A *termname* can not be the same as any keyword listed under 'format'.
2. *termname* values can be used as search values on the TRACE command directly. No abbreviations for *termname* are allowed.

Notes about TRACE Entry Definitions

1. Data from Trace Entry definitions are used to fill the output line to the length specified by 'VMDTSET LINESIZE nnn'. Data that does not fit is spilled by separate line in the trace entry definition. For the following example, even though the string 'by' might fit on a given output line, if the whole 'by' and module won't fit, then all of the data will be deferred to the next output line.

   ```
   'by'           module    ttedata4    4
   ```

2. An example:

   0100 control

   - this is the definition of the 0100 trace entry. It will be found by a TRACE TYPE CONTROL command.

     ```
     extcode   ttedata2b2  2
     ```

   - there is no string
   - 'extcode' is a term (which happens to be defined in a table)

- the data starts at the displacement 'ttedata2b2'. These tags refer to the field names and the byte within the field. See HCPTTEBK copy.
- the length of the data is 2.
- Some terms have look up tables in the front of the dump. 'extcode' happens to be one of them. Here's the definition of EXTCODE:

  term extcode

  ```
  1003 'Tod-Clock Sync Ck'
  1004 'Clock Comp'
  1005 'CPU Timer'
  1200 'Malfcn Alert'
  1201 'Emerg Signal'
  1202 'Ext Call'
  2401 'Service Signal'
  ```

  What this means is if the data found at displacement ttedata2b2 X'16' for a length of 2 has the value of '1005', then the string 'CPU Timer' will be displayed.

```
*   not used       hex       ttedata0    4
```

This is simply a comment to indicate that the 0 field is not used.

```
  'ExtInt at'     module    ttedata4    4
```

'ExtInt at' is the string that will be displayed first when this part of the 0100 trace entry is processed.

'module' indicates that the value from the trace entry should be displayed as an entry point or module+displacement.

'ttedata4' is the displacement to the field in the trace entry where this data is found.

4 is the length of the field in the trace entry.

```
  'opsw'          hex       ttedata3    8
```

Similar. 'opsw' is simply a string that will be displayed.

'hex' is a keyword that indicates the data should be displayed in hex.

'ttedata3' is the name of the field where this data resides.

'8' is the length of the field.

Examples

Here's a real example of a type 0100 entry:

```
>>> d r7DC9F180.20
_7DC9F180 +0000 7400DDFA 94156786 00000100 00000000 *...³m..f........*
_7DC9F190 +0010 40C5E7E3 00001201 070E4000 80000000 * EXT...... .....*
```

Here's the definition (from above)

```
0100 control
                   extcode   ttedata2b2  2
   'ExtInt at'     module    ttedata4    4
   'opsw'          hex       ttedata3    8
```

And here's what came out the bottom:

```
>>> trace for 1 from 7DC9F180
7DC9F180 07 0100 Emerg Signal ExtInt at 80000000
                 opsw 070E4000_80000000
```

The reason the 'opsw' data was put on the second line is that the line length is set pretty short. With a longer line, the output of the new trace will fill more of the line:

```
>>> dts lin 200
complete
>>> trace for 1 from 7DC9F180
7DC9F180 07 0100 Emerg Signal ExtInt at 80000000 opsw 070E4000_80000000
```

The following example is more detailed:

```
*-------------------------------------
* 0600 - obtain free storage
*
0600 free
    'Obtain'       dec       ttedata1    4
    'dw'           pchr      ttedata0b1  3
    'at'           address   ttedata2    4
    'by'           module    ttedata4    4
    'vmdbk'        vmdbk     ttedata3    4
                   user      ttedata3    4

-- 'free' is a TYPE that can be searched on.
-- 'dec' means to format the data as a decimal number.
-- 'pchr' says to format a character string and put it in parens.
-- 'address' is a TERM that can be searched on.
-- 'module' says to convert the address to a module name.
-- 'vmdbk' is a TERM that can be searched on.
-- 'user' means to convert the VMDBK address to the user ID associated
   with that VMDBK. 'user' can also be searched on by specifying
   'USER userid' on the TRACE command.

>>> d r7F6D1D00.20
_7F6D1D00 +0000 7400DDFA 9401FF85 00000600 4CC7E2F1 *...³m..e....<GS1*
_7F6D1D10 +0010 0000000D 7BE8D6F0 7CD27000 84975470 *....#YO0@K..dp..*

>>> trace for 1 code 0600
7F6D1D00 00 0600 Obtain 13 dw (GS1) at 7BE8D6F0 by HCPLNK+470 vmdbk 7CD27000

You can search on the 'address' and find the same data:

   >>> trace for 3 address 7E43F698 merge
   7F6D1DE0 00 0700 Release 24240 dw (LKX) at 7E43F698 by HCPLNK+426
                    vmdbk 7CD27000 TGREER1
   7F6D1DA0 00 0700 Release 37 dw (LKX) at 7E43F698 by HCPLNK+426
                    vmdbk 7CD27000 TGREER1
   7F6D1D00 00 0600 Obtain 13 dw (GS1) at 7BE8D6F0 by HCPLNK+470
                    vmdbk 7CD27000 TGREER1
Or you can search by the VMDBK address:

>>> trace for 8 vmdbk 7CD27000 merge
7F6D1DE0 00 0700 Release 24240 dw (LKX) at 7E43F698 by HCPLNK+426
                 vmdbk 7CD27000 TGREER1
7F6D1DA0 00 0700 Release 37 dw (LKX) at 7E43F698 by HCPLNK+426
                 vmdbk 7CD27000 TGREER1
7F6D1D40 00 0700 Release 13 dw (GS1) at 7BE8D6F0 by HCPREC+620
                 vmdbk 7CD27000 TGREER1
7F6D1D00 00 0600 Obtain 13 dw (GS1) at 7BE8D6F0 by HCPLNK+470
                 vmdbk 7CD27000 TGREER1
7F6D1C60 00 0700 Release 5 dw (LCK) at 7E501C48 by HCPLOC+7C
                 vmdbk 7CD27000 TGREER1
7F6D1B80 00 3700 Stk Wk Bits 00000002 vmdbk 7CD27000 TGREER1
                 by HCPVMN+156 vmdstate 4D vmdslist 37 mod-id KW
7F6D1B60 00 0600 Obtain 7 dw (MCV) at 7BE8D770 by HCPVMN+2F4
                 vmdbk 7CD27000 TGREER1
7F6D1B40 00 0600 Obtain 2 dw (CRW) at 7EE50B48 by HCPVMN+2C0
                 vmdbk 7CD27000 TGREER1
```

A Term Definition List for the term 'extcode':

```
term extcode
      1003 'Tod-Clock Sync Ck'
      1004 'Clock Comp'
      1005 'CPU Timer'
      1200 'Malfcn Alert'
      1201 'Emerg Signal'
      1202 'Ext Call'
      2401 'Service Signal'
```

Trace entry definition for type 8100:

```
8100 control
                     extcode   ttegdat1b6  2
       'ExtInt from cpu' hex    ttegdat1b4  2
       'parm'         hex       ttegdat1    4
       'opsw'         hex       ttegdat2   16
```

Raw data from a trace entry to be formatted:

```
>>> d r7F800400.40
   _7F800400 +0000 7580BE3F 13496CC5 FE200000 00008100 *......%E......a.
   _7F800410 +0010 40C5E7E3 00000000 0001D000 00051202 * EXT......}.....
   _7F800420 +0020 07064000 80000000 00000000 00000000 *.. ............
   _7F800430 +0030 00000000 00000000 00000000 00000000 *................
```

Formatted output:

```
>>> trace for 1 code 8100
   7F800400 11:30:52 Ext Call ExtInt from cpu 0005 parm 0001D000
                     opsw 07064000 80000000 00000000 00000000
```

Explanation:

- The data for 'extcode' was fetched from ttegdat1b6 for 2 bytes, which is displacement 1E in the raw trace entry.
- The data was found to be 1202.
- The 1202 was used to search the Term Definition List for 'extcode'.
- The logic found that the string 'Ext Call' is associated with an extcode of 1202.
- 'Ext Call' was inserted in the output to be displayed.

Writing Trace Macros

The primitive operations defined for use in Trace Defintion Files were designed to allow you to format most of the data that is commonly found in trace entries. But sometimes the data is more complicated than the primitives allow, such when the format of the data depend on some other data in the trace entry itself. The solution is to write a trace macro to apply logic to the formatting.

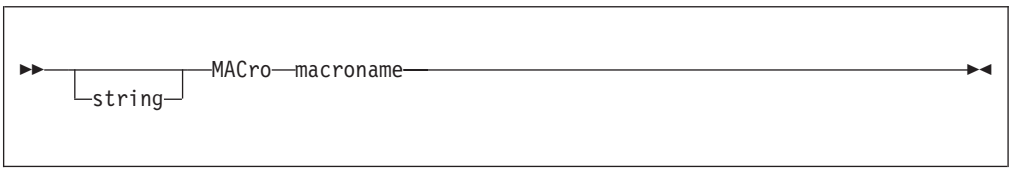
A VM Dump Tool trace macro can be written to handle formatting a whole trace entry or part of one. The built in primitives have been written to be as efficient as possible, so it often makes the most sense to have the macro format only the parts that require decisions.

To invoke a macro, specify the 'macro' primitive in a Trace Definition File.

Purpose

The MACRO statement is used in a VM Dump Tool Trace Definition File to specify that a macro is to be invoked at that point during the formatting for a given trace entry.

Syntax



Operands

string
specifies an optional string that will be added to the TRACE output before the output produced by the macro.

macroname
is the name of the VM Dump Tool TRACE macro to be invoked. The file type of the macro must be VMDTTRC. It may be on any disk or SFS directory in the search order.

Input

When the macro is given control, three parameters are included:

- Interface Level
- Trace Entry Code
- Parm List

Interface Level

The Interface Level indicates the level of the VM Dump Trace Macro interface. At this point, it is decimal 102. If the interface ever changes significantly, then the VM Dump Tool will pass in a different value. The macro should check this value and terminate if it does not recognize the level number.

The intent is to change this number only when something is changed in the specified interface. Extensions to use reserved fields are not likely to change the interface level number since they will usually not change anything that was defined in the past.

Trace Entry Code

The trace entry code is a four-digit readable hexadecimal value which tells the trace macro the trace entry code of the trace entry that is to be formatted. The macro can use this to make decisions about where data is to be obtained, or how data is to be formatted if it is invoked for different trace entry codes and the data or format varies.

Parameter List

A parameter list is passed to the trace macro as the third input which includes the whole trace entry, the host real address of the trace entry in the dump, the CPU number and some status.

The data is in readable hex format (2 characters of data in the parm list for every byte of original data) and is 192 bytes in length.

There are two slightly different formats depending on the type of the trace entry being formatted. The displacements are origin 1 and in decimal since that is how you'll need to reference this data from REXX or other EXEC language.

The following is for a type-74 entry (32 bytes in length):

1	CPU#	flag	reserved
17	trace entry real address		
33	standard part of type-74 trace entry		
49	standard part		ttedata0
65	ttedata1		tt3data2
81	ttedata3		ttedata4
97	not used		
113	not used		
129	not used		
145	reserved		
161	reserved		
177	reserved		

A type-75 entry (64 bytes in length) has the following format:

1	CPU#	flag	reserved
17	trace entry real address		
33	standard part of trace entry, 1 of 2		
49	standard part of trace entry, 2 of 2		
65	ttedata0		
81	ttedata1		
97	ttedata2		
113	ttedata3		
129	ttedata4		
145	ttedata5		
161	reserved		
177	reserved		

Where:

CPU# indicates which CPU this trace entry is associated with.

flag is set to 00 if the BITS setting has been defaulted or overtly set to OFF. It is set to 01 if BITS is on so bit values should be displayed.

trace entry real address
is filled in with the 16-digit hexadecimal real address of where this trace entry is located in host real storage.

standard parts
are filled in with the parts of the trace entry that are fixed in meaning. It is not expected that this will be used by a macro very much, but if a macro wants to look at this data, it is there to find.

ttexxxxx fields
are filled in with the content of the CP-specified portions of the trace entry.

not used
means that this area is not filled in when the macro is called. The contents are unpredictable for a 74-type trace entry.

reserved
are filled with character 0.

TRACEOUT Command

The normal way for VM Dump Tool macros to generate output data is to place that data in the CMS Program Stack. However, the TRACE command works a little differently. Instead of simply accumulating data lines and adding them to the output stream, it fills each output line as full as it can and spills the rest. For the output of the macro to be included in this process, the macro must use the TRACEOUT command.

Purpose

Display some data from as part of the output of a TRACE command.

Operands

output
is any sequence of numbers and letters to be added to the output.

Usage Notes

1. All of the output specified by one TRACEOUT command is put on the output line, or spilled, as a unit. If the trace macro produces data that you wish to be allowed to spill separately, then you should specify the separate parts of the output in multiple invocations of TRACEOUT.
2. The only restriction on the amount of data that can be specified on TRACEOUT is that the VM Dump Tool limits one line of input to 512 characters.
3. The efficiency of trace macros depends on a number of variables including how much work is done by the macro and how often it is invoked. To reduce the macro-related overhead, you might consider compiling your macro, and/or EXECLOAD to load it into CMS storage before invoking the VM Dump Tool. (Don't forget to EXECDROP it after you leave).

4. The return code from a VM Dump Tool trace macro has no meaning and is ignored.
5. The actual content of each trace entry can be found in the *z/VM: Diagnosis Guide*.

Annotated Trace Macro Example

As an example, let's look at the HCQT3310 macro that is shipped with the VM Dump Tool to help format the 3310 trace entry.

Here is the data for the 3310 trace entry in the HCQTRACE VMDTDATA Trace Definition File:

```
3310 stack control
     'Unstack CPEBK'
     'at'           address   ttedata3    4
                    macro     hcqt3310
     'vmdbk'        vmdbk     ttedata2    4
                    user      ttedata2    4
     'cpexschc'     hex       ttedata1b3  1
* handled by the macro ...    ttedata4    4
     byte           cpexschc  ttedata1b3
```

In this example, when HCQT3310 VMDTTRC is invoked, two character strings and four bytes of hexadecimal data have already been added to the output.

The TTEDATA4 data in a 3310 trace entry can be one of two things depending on status found in TTEDATA1. The low order byte of TTEDATA1 is the CPEXSCH byte from the CPEBK being unstacked. If either the CPEXSKCR bit (x'40'), or the CPEXRTNF bit (x'10') is on, then TTEDATA4 is the return address. Otherwise TTEDATA4 is a return code.

The following is a copy of HCQT3310 VMDTTRC. Line numbers have been added on the left which refer to explanations below.

```
   /*--------------------------------------------------------------------*/
   /* HCQT3310 VMDTTRC - Partial formatter for 3310 trace entry          */
   /*   Part of the VM Dump Tool package                                 */
   /*--------------------------------------------------------------------*/
   /*   COPYRIGHT ...                                                    */
   /*                                                                    */
   /*--------------------------------------------------------------------*/
 1  Trace 'Off'
 2  Numeric Digits 12
 3  Address VMDUMPTL

   /*--------------------------------------------------------------------*/
   /* Get input parms, check interface level                             */
   /*--------------------------------------------------------------------*/
 4  Arg ilvl trcent parm .
 5  If ilvl <> '102' Then Signal LVLERROR

   /*--------------------------------------------------------------------*/
   /* Load variables from VM Dump Tool parm list                         */
   /*--------------------------------------------------------------------*/
 6     Parse Var parm . 57 ttedata0 65 ttedata1 73 ttedata2,
 7        81 ttedata3 89 ttedata4 97 .

   /*--------------------------------------------------------------------*/
   /* Get bits from CPEXSCHC                                             */
   /*--------------------------------------------------------------------*/
 8  bits = X2B(Substr(ttedata1,7,2))
 9  rtrn1 = 2                        /* 40 CPEXSKCR */
```

```
10  rtrn2 = 4                          /* 10 CPEXRTNF */

   /*--------------------------------------------------------------------*/
   /* If either RETURN bit is on, format RC as a return code             */
   /*--------------------------------------------------------------------*/
11  If Substr(bits,rtrn1,1) = 1 | ,
12        Substr(bits,rtrn2,1) = 1 Then Do
13     rc = X2D(ttedata4)
14     Select
15        When rc < 0    Then new = 'rc<0'
16        When rc < 4096 Then new = 'rc='rc
17        Otherwise           new = 'rc>FFF'
18        End /* select */
19     End /* if return */

   /*--------------------------------------------------------------------*/
   /* Not a return, R15 must be the 'to' address                         */
   /*--------------------------------------------------------------------*/
20  Else Do
21     'EXTRACT MAPA from' ttedata4 'to modname' /* get module name */
22     new = 'to' Strip(modname)      /* form new output */
23     End

   /*--------------------------------------------------------------------*/
   /* Add new info to existing line                                      */
   /*--------------------------------------------------------------------*/
24  'TRACEOUT' new

25  Exit 0

   /*--------------------------------------------------------------------*/
   /* LVLERROR Routine - Error found in the macro interface level        */
   /*--------------------------------------------------------------------*/
26  LVLERROR:
27     msgid   = 'HCQX33'
28     macname = 'HCQT3310'
29     Address COMMAND 'CP MSG *',
30        msgid'145E macro interface level mismatch in' macname':' ilvl
31     Exit 0
```

Lines 1-3 - normal macro linkage

```
1  Trace 'Off'
2  Numeric Digits 12
3  Address VMDUMPTL
```

Line 1:
REXX statement that turns off tracing of errors

Line 2:
Needed so the macro can compute with 32-bit hex values

Line 3:
Make the VM Dump Tool the default environment

Lines 4-5, 26-31, get inputs, check interface level

```
4  Arg ilvl trcent parm .
5  If ilvl <> '102' Then Signal LVLERROR

26 LVLERROR:
27    msgid   = 'HCQX33'
28    macname = 'HCQT3310'
```

```
29      Address COMMAND 'CP MSG *',
30          msgid'145E macro interface level mismatch in' macname':' ilvl
31      Exit 0
```

Line 4:
reads the parameters passed in to the macro into 3 variables. The interface level is placed in variable *ilvl*, the four-digit trace entry code is placed in variable *trcent*, and the parameter list is placed in variable *parm*.

Line 5:
This test checks to see if the interface level is one that we recognize. If it is not, control goes to the statement labelled LVLERROR:

Line 26:
This merely defines the label which is to be branched to.

Lines 6-7, break apart the parm list

```
6       Parse Var parm . 57 ttedata0 65 ttedata1 73 ttedata2,
7           81 ttedata3 89 ttedata4 97 .
```

Lines 6-7:
This is one REXX PARSE statement to break up the parameter list into parts and load the parts into the indicated variables. Literally, '57 ttedata0 65' says to load the contents of displacement 57 through 64 (because the next field starts in 65) into the variable named ttedata0.

Lines 8-10, capture the 2 significant bits

```
 8  bits = X2B(Substr(ttedata1,7,2))
 9  rtrn1 = 2                          /* 40 CPEXSKCR */
10  rtrn2 = 4                          /* 10 CPEXRTNF */
```

Line 8:
picks up the low order byte of TTEDATA1, converts to bits.

Line 9:
is loaded with the position in the byte of the first bit.

Line 10:
is loaded with the position in the byte of the second bit.

Lines 11-19, Handle the case where it's a return code

```
11  If Substr(bits,rtrn1,1) = 1 | ,
12         Substr(bits,rtrn2,1) = 1 Then Do
13      rc = X2D(ttedata4)
14      Select
15          When rc < 0      Then new = 'rc<0'
16          When rc < 4096 Then new = 'rc='rc
17          Otherwise              new = 'rc>FFF'
18          End /* select */
19      End /* if return */
```

Line 11:
is testing if the first bit is turned on.

Line 12:
is testing if the second bit it turned on.

Line 13:
gets control if either of these bits is turned on. It loads the TTEDATA4 value into the variable RC.

Line 14-18:
load the variable NEW with the string appropriate for the return code in RC.

Lines 20-23, Convert address to Module Name

```
20  Else Do
21     'EXTRACT MAPA from' ttedata4 'to modname' /* get module name */
22     new = 'to' Strip(modname)        /* form new output */
23     End
```

Line 20:
This line gets control if the test in lines 11 and 12 is not true.

Line 21:
issues the EXTRACT command to convert the address found in variable TTEDATA4 to a CP module name.

Line 22:
EXTRACT can return trailing blanks. Invoking the Strip function removes the extra blanks.

Line 23:
This simply delineates the end of this DO group.

Line 24: Add the output to the dump session.

```
24  'TRACEOUT' new
```

Line 24:
This invokes the TRACEOUT command to add the appropriate output (whatever was set up in lines 15-17 or line 22) to the output session.

Line 25: Return to the VM Dump Tool

```
25  Exit 0
```

Line 25:
This simply exits the macro and returns RC=0 to the VM Dump Tool TRACE handler which will go on to the next part of the Trace Definition for this trace entry.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The

sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Privacy Policy Considerations

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 IBM Online Privacy Policy at http://www.ibm.com/privacy and the IBM Online Privacy Statement at http://www.ibm.com/privacy/details, in particular the section entitled “Cookies, Web Beacons and Other Technologies”, and the IBM Software Products and Software-as-a-Service Privacy Statement at http://www.ibm.com/software/info/product-privacy.

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 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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at IBM copyright and trademark information - United States (www.ibm.com/legal/us/en/copytrade.shtml).

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Glossary

For a list of z/VM terms and their definitions, see *z/VM: Glossary*.

The z/VM glossary is also available through the online z/VM HELP Facility, if HELP files are installed on your z/VM system. For example, to display the definition of the term "dedicated device", issue the following HELP command:

```
help glossary dedicated device
```

While you are in the glossary help file, you can do additional searches:

- To display the definition of a new term, type a new HELP command on the command line:

  ```
  help glossary newterm
  ```

  This command opens a new help file inside the previous help file. You can repeat this process many times. The status area in the lower right corner of the screen shows how many help files you have open. To close the current file, press the Quit key (PF3/F3). To exit from the HELP Facility, press the Return key (PF4/F4).
- To search for a word, phrase, or character string, type it on the command line and press the Clocate key (PF5/F5). To find other occurrences, press the key multiple times.

  The Clocate function searches from the current location to the end of the file. It does not wrap. To search the whole file, press the Top key (PF2/F2) to go to the top of the file before using Clocate.

Bibliography

See the following publications for additional information about z/VM. For abstracts of the z/VM publications, see *z/VM: General Information*, GC24-6193

Where to Get z/VM Information

z/VM product information is available from the following sources:

- IBM Knowledge Center z/VM welcome page (www.ibm.com/support/knowledgecenter/SSB27U/welcome)
- IBM Publications Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss)
- *IBM Online Library: z/VM Collection*, SK5T-7054
- IBM: z/VM Internet Library (www.ibm.com/vm/library/)

z/VM Base Library

Overview

- *z/VM: General Information*, GC24-6193
- *z/VM: Glossary*, GC24-6195
- *z/VM: License Information*, GC24-6200

Installation, Migration, and Service

- *z/VM: Installation Guide*, GC24-6246
- *z/VM: Migration Guide*, GC24-6201
- *z/VM: Service Guide*, GC24-6247
- *z/VM: VMSES/E Introduction and Reference*, GC24-6243

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6167
- *z/VM: CMS Planning and Administration*, SC24-6171
- *z/VM: Connectivity*, SC24-6174
- *z/VM: CP Planning and Administration*, SC24-6178
- *z/VM: Enabling z/VM for OpenStack (Support for OpenStack Icehouse Release)*, SC24-6248
- *z/VM: Enabling z/VM for OpenStack (Support for OpenStack Juno Release)*, SC24-6249
- *z/VM: Enabling z/VM for OpenStack (Support for OpenStack Kilo Release)*, SC24-6250
- *z/VM: Getting Started with Linux on System z*, SC24-6194
- *z/VM: Group Control System*, SC24-6196
- *z/VM: I/O Configuration*, SC24-6198
- *z/VM: Running Guest Operating Systems*, SC24-6228
- *z/VM: Saved Segments Planning and Administration*, SC24-6229
- *z/VM: Secure Configuration Guide*, SC24-6230
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6236
- *z/VM: TCP/IP Planning and Customization*, SC24-6238
- *z/OS and z/VM: Hardware Configuration Manager User's Guide*, SC33-7989

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6176
- *z/VM: Performance*, SC24-6208

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6166
- *z/VM: CMS Pipelines Reference*, SC24-6169
- *z/VM: CMS Pipelines User's Guide*, SC24-6170
- *z/VM: CMS Primer*, SC24-6172
- *z/VM: CMS User's Guide*, SC24-6173
- *z/VM: CP Commands and Utilities Reference*, SC24-6175
- *z/VM: System Operation*, SC24-6233
- *z/VM: TCP/IP User's Guide*, SC24-6240
- *z/VM: Virtual Machine Operation*, SC24-6241
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6244
- *z/VM: XEDIT User's Guide*, SC24-6245

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6162
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6163
- *z/VM: CMS Application Multitasking*, SC24-6164
- *z/VM: CMS Callable Services Reference*, SC24-6165

- *z/VM: CMS Macros and Functions Reference*, SC24-6168
- *z/VM: CP Programming Services*, SC24-6179
- *z/VM: CPI Communications User's Guide*, SC24-6180
- *z/VM: Enterprise Systems Architecture/Extended Configuration Principles of Operation*, SC24-6192
- *z/VM: Language Environment User's Guide*, SC24-6199
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6202
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6203
- *z/VM: OpenExtensions Commands Reference*, SC24-6204
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6205
- *z/VM: OpenExtensions User's Guide*, SC24-6206
- *z/VM: Program Management Binder for CMS*, SC24-6211
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6220
- *z/VM: REXX/VM Reference*, SC24-6221
- *z/VM: REXX/VM User's Guide*, SC24-6222
- *z/VM: Systems Management Application Programming*, SC24-6234
- *z/VM: TCP/IP Programmer's Reference*, SC24-6239
- *Common Programming Interface Communications Reference*, SC26-4399
- *Common Programming Interface Resource Recovery Reference*, SC31-6821
- *z/OS: IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148
- *z/OS: Language Environment Concepts Guide*, SA22-7567
- *z/OS: Language Environment Debugging Guide*, GA22-7560
- *z/OS: Language Environment Programming Guide*, SA22-7561
- *z/OS: Language Environment Programming Reference*, SA22-7562
- *z/OS: Language Environment Run-Time Messages*, SA22-7566
- *z/OS: Language Environment Writing Interlanguage Communication Applications*, SA22-7563
- *z/OS MVS Program Management: Advanced Facilities*, SA22-7644
- *z/OS MVS Program Management: User's Guide and Reference*, SA22-7643

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6161
- *z/VM: CP Messages and Codes*, GC24-6177
- *z/VM: Diagnosis Guide*, GC24-6187
- *z/VM: Dump Viewing Facility*, GC24-6191
- *z/VM: Other Components Messages and Codes*, GC24-6207
- *z/VM: TCP/IP Diagnosis Guide*, GC24-6235
- *z/VM: TCP/IP Messages and Codes*, GC24-6237
- *z/VM: VM Dump Tool*, GC24-6242
- *z/OS and z/VM: Hardware Configuration Definition Messages*, SC33-7986

z/VM Facilities and Features

Data Facility Storage Management Subsystem for VM

- *z/VM: DFSMS/VM Customization*, SC24-6181
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6182
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6183
- *z/VM: DFSMS/VM Planning Guide*, SC24-6184
- *z/VM: DFSMS/VM Removable Media Services*, SC24-6185
- *z/VM: DFSMS/VM Storage Administration*, SC24-6186

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6188
- *z/VM: Directory Maintenance Facility Messages*, GC24-6189
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6190

Open Systems Adapter/Support Facility

- *z Systems: Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935
- *System z9 and eServer zSeries 890 and 990: Open Systems Adapter-Express Integrated Console Controller User's Guide*, SA22-7990
- *System z: Open Systems Adapter-Express Integrated Console Controller 3215 Support*, SA23-2247

- *System z10: Open Systems Adapter-Express3 Integrated Console Controller Dual-Port User's Guide*, SA23-2266

Performance Toolkit for VM

- *z/VM: Performance Toolkit Guide*, SC24-6209
- *z/VM: Performance Toolkit Reference*, SC24-6210

RACF Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6212
- *z/VM: RACF Security Server Command Language Reference*, SC24-6213
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6214
- *z/VM: RACF Security Server General User's Guide*, SC24-6215
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6216
- *z/VM: RACF Security Server Messages and Codes*, GC24-6217
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6218
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6219
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6231

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6223
- *z/VM: RSCS Networking Exit Customization*, SC24-6224
- *z/VM: RSCS Networking Messages and Codes*, GC24-6225
- *z/VM: RSCS Networking Operation and Use*, SC24-6226
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6227

Prerequisite Products

Device Support Facilities

- *Device Support Facilities: User's Guide and Reference*, GC35-0033

Environmental Record Editing and Printing Program

- *Environmental Record Editing and Printing Program (EREP): Reference*, GC35-0152
- *Environmental Record Editing and Printing Program (EREP): User's Guide*, GC35-0151

Index

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Product Number: 5741-A07

Printed in USA

GC24-6242-03