To the Graduate Council:

I am submitting herewith a thesis written by Randy G. Dotson entitled "Disk Management of the VAX/VMS Disk Subsystem." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

K. R. Kimble, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The Univerisy of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Randy H. Dotson

Date May 10, 1988

DISK MANAGEMENT OF THE VAX/VMS DISK SUBSYSTEM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Randy G. Dotson

June 1988

DEDICATION

To my wife, Andrea, and my daughter, Ashley: You are the joys of my life. Thank you for your support, patience, and confidence in my endeavors.

ACKNOWLEDGMENTS

In the course of thesis research, a number of scholars have assisted me generously. In particular I am indebted to Professors Kenneth Kimble, Seung-Chul Lee, and Wilbur Armstrong, who served as my director and committee respectively. With their consultation, criticism, and encouragement, I was able to achieve results otherwise impossible. Whatever errors remain in my work, however, are solely my responsibility.

ABSTRACT

The disk storage subsystem on a VAX/VMS system plays an important role in overall system performance. All VMS systems, from MicroVaxes to 8800s, depend on disk media to provide accessibility to the users' data. Since all computers are truly information processors, the disk storage subsystem in a modern computing environment can be very costly if it is not properly managed. Disk management is therefore a proper concern for all system managers, however, it is all to often a tedious, time-consuming management task.

Efficient disk management requires that a disk's major characteristics be monitored and made readily available to the disk system manager. Currently, many VMS utilities must be used to gather this important information. It is the goal of the author to provide a single source utility to simplify the burden associated with disk management. If disk system management is sufficiently understood and its limitations are known, proper disk system management can be a very interesting and rewarding pursuit.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

MANAGING THE VMS DISK SUBSYSTEM

Managing the VAX/VMS disk storage subsystem involves optimizing the amount of time it takes to complete a disk operation. Currently, there are many VMS utilities that aid the system manager in disk management. These utilities provide a computer educated user with enough capabilities that optimal disk system performance can be achieved. However, an uneducated user may encounter many problems while using these same utilities. This unfortunate consequence occurs because of a design flaw in the current VMS system management architecture. This thesis is an attempt to provide the non-professional, unsophisticated user with a utility to ease the disk management burden.

1.1 VMS SYSTEM MANAGEMENT ARCHITECTURE

The VMS operating system has become a very useful operating system since its initial development, more than ten years ago. Many powerful features, capabilities, and enhancements have made VMS easier to use and easier to enjoy. The VMS designers have consistently provided new

commands and utilities for the betterment of their user community. For example, VMS offers a wide range of products to support different programming efforts. Everything from language sensitive editors, to efficient debuggers, to complex code analyzers has been developed and is currently available. Not to mention the programming power provided from VMS' standard run-time libraries and system services. As demonstrated by this example, VMS designers are more than willing to make the user's life with VMS as productive as possible. In contrast, the underlying architecture associated with system management has changed very little since 1978.

Not to be misunderstood, VMS has added several commands and utilities to aid the system manager. For the most part, these utilities were very much needed and very much applauded when they were introduced. The problem with the current VMS system management design is not due to the number of utilities, but rather the problem stems from an assumption made ten years ago by the original VMS designers. This assumption states that a VMS system will always be managed by a dedicated, professional, computer person, capable of understanding a complex operating system. This assumption has produced a major limitation to the VMS system management architecture.

1.1.1 Background Of Current Architecture

When VMS was designed in 1978, only one hardware system was available; the VAX 11/780. At that time, the assumption of a professional computer person being tasked with the system management responsibilities was very accurate. The cost of a VAX 11/780 in 1978

approached the half-million dollar price range. With a CPU as powerful as the VAX 11/780, a system manager was considered a necessity. However, the current number of VAX configurations, both upward and downward from the VAX 11/780, has strained this original assumption.

To fully understand the current VMS system management architecture, the reader must first understand what is required from a system manager. To properly manage a VMS system, it is imperative that the system manager be well versed in the terms and concepts of VAX/VMS. As mentioned before, VMS is a very complex operating system. To manage it in an efficient manner, its terminology must be understood. If the system manager lacks such an understanding, many unnecessary problems may be encountered. However, once these concepts have been mastered, the system manager is equipped to put this knowledge to work. Applying this knowledge is often easier said than done as will be demonstrated in the next section.

1.1.2 Problem Definition

VMS offers many utilities that are designed to ease the system manager's task. Almost every aspect of system management has been addressed by some existing VMS utility. However, many of these utilities have very different interfaces which increases the learning curve for a system manager. This lack of consistency and order in current utilities creates an environment that requires a concentrated effort from the system manager. That is, the system manager is required to be fluent in the operation of many VMS utilities in order to monitor any one resource (such as CPU, memory, or disk). Again,

this problem relates to the original design assumption that a dedicated professional will manage a VMS system.

Today, a full spectrum of VAX machines, each of which support the same VMS operating system, are available. In particular, the MicroVAX II is very well accepted as a dedicated system for applications requiring engineering workstations, real-time data acquisition, and computer aided design and development. The proliferation of MicroVAX II systems has magnified the problems associated with the current VMS system management architecture. This situation has developed because many MicroVAX II systems are not managed by computer professionals. Rather, the management responsibilities are usually assigned to someone with little or no computer education or experience. Also, this person usually has other job-related responsibilities than just those of system management. Needless to say, management of systems where the system manager is not truly qualified can result in less than optimal system performance.

The VMS designers are aware of these current limitations and are taking the appropriate steps to provide a fully integrated approach to system management. These designers are currently defining a new internal architecture for system management. This architecture will provide a modular base for all system management efforts over the next several releases of VMS. The initial implementation, currently called SYSMAN, is scheduled for release with VMS V5.0 in mid-1988. However, this initial attempt will focus only on high-end VAXcluster management, leaving low-end small VAX users groping in the dark, waiting for later development enhancements

1.2 INTRODUCTION TO DISKMGR

Disk Manager (DISKMGR) is a system management utility for managing the VAX/VMS disk storage subsystem. It is a utility that will combat many of the problems effecting the current VMS system management architecture. For example, DISKMGR provides interactive help text to educate new system managers on disk related concepts and terms. Also, DISKMGR is a single source utility. By using only DISKMGR, reasonably efficient disk management can be expected. Furthermore, DISKMGR allows its user to learn disk management while actually performing necessary disk management functions. All in all, DISKMGR provides a consistent user interface that does not require a computer professional to operate. This utility will hopefully provide relief from the everyday frustrations associated with the current VMS system management architecture.

CHAPTER 2

DISK MANAGEMENT WITH VAX/VMS

This chapter provides a detailed look at the concepts involved in the VAX/VMS disk storage subsystem. It also details why the disk storage subsystem has a great impact on overall system performance and discusses some limiting factors involved with disk optimization.

2.1 THE HISTORY OF FILE STORAGE MEDIA

It seems no matter how many advances are made in disk media technology, the volume of information to be saved is always larger than the capacity of the system. Large companies that once could store all their data in desk drawers and filing cabinets, now depend on disk media for retrieving data quickly, reliably, and efficiently. The storage of a company's data on paper has become very impractical for even the smallest of companies. Over the past ten years, many advances have been registered in the capabilities and capacities of our physical storage media. The following paragraphs retrace our footprints of the past.

At first, card decks provided a convenient means of storing data. As a storage medium, cards have some distinct advantages. For one thing, cards are easy to manipulate and are generally quite inexpensive. However, cards become worn, require physical handling, and are relatively slow to process. Also, anyone who has ever dropped a deck of bulky cards and had to resequence them, can truly appreciate the capabilities of modern technology.

Next came the introduction of magnetic tape as a storage medium. Magnetic tape eliminated some of the disadvantages of cards as it offered both input and output capabilities, as well as, virtually unlimited storage when compared to a card file. However, magnetic tape media is limitied as a storage medium because, like the card file, it allows only sequential file access. Comparing either of these storage medias with today's disk capabilities is like comparing the horse-and-buggy to the airplane. This is not to imply that cards and tapes are not useful. There are very few computer systems in the world today without at least one tape drive or card reader attached to it. Although tapes and cards are slow, they still have a place in the computer room (Digital, "Guide to VAX/VMS Disk and Magnetic Tape Operations").

Today, disk media technology is providing increased capabilities for the manipulation of all types of information. The amount and type of information that can be retained in storage has changed significantly from the first electronic computers. These first computers were capable of storing only a few numbers, whereas today's modern devices are capable of storing trillions of characters. As the

availability of large amounts of information becomes more of a necessity than a luxury, disk performance will become a more crucial cost factor in evaluating the effectiveness of a computer system.

2.2 BASIC VAX/VMS DISK CONCEPTS

VAX/VMS disk files reside on disk media that has been formatted using the Files-11 On-Disk Structure organization. The organization of files in this manner is maintained by the VAX/VMS file system which is responsible for all disk access. This section describes the Files-11 On-Disk Structure levels and defines the terminology necessary to understand the structure.

A disk's smallest addressable entity is known as a block. A block is defined as 512 eight-bit bytes, each of which are contiguous on the disk. Blocks are logically grouped into clusters. A cluster may consist of one or more blocks and is the basic unit of exchange used for disk space allocation. A file with only one character requires at least one cluster of blocks for storage. The disk system manager is responsible for determining how many blocks form a cluster, based on the type of operations to be performed on the disk. In general, if data files are small, the cluster size should be small so as to conserve valuable disk space.

To further build upon our terminology, an extent is a series of one or more contiguous clusters allocated to a particular file. An extent can contain all or part of a file. If sufficient contiguous area is available, the entire file is allocated as a single extent. Fragmentation occurs when a file consists of multiple extents located

in separate areas of the disk. Figure 2-1 shows a typical file layout with a cluster size equal to two.

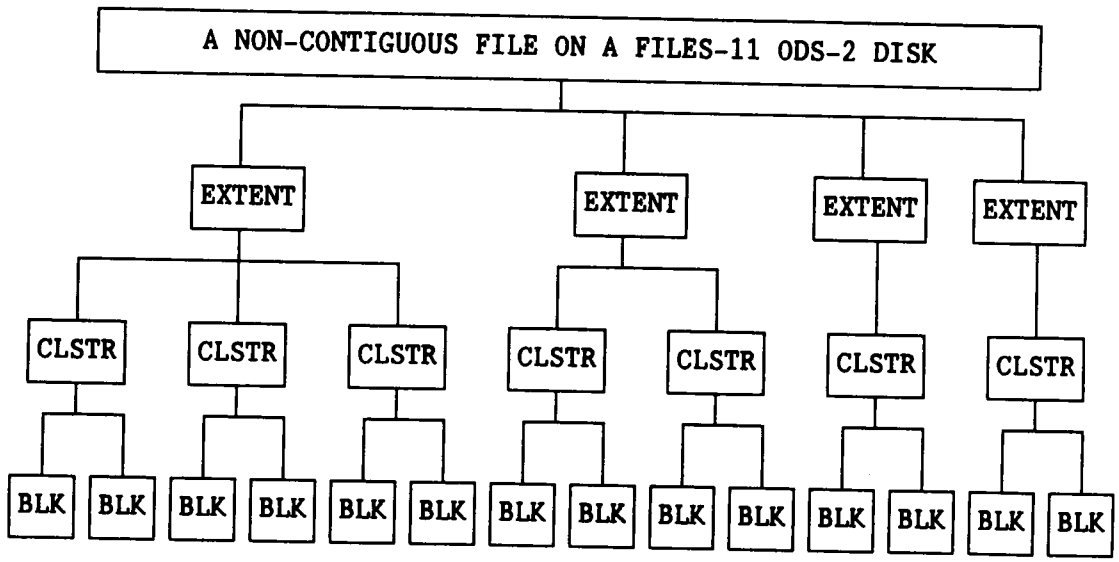


Figure 2-1. File Layout On Disk

The smallest unit of the physical medium is known as a sector. For most Files-11 disks, a sector is equivalent to a block. Similarly, a track is a collection of sectors on one recording surface of the disk. A cylinder consists of all the tracks at the same radius on all recording surfaces. Related data should be stored in the same cylinder to minimize the movement of read/write heads during data seeks. More information concerning the hardware of a disk will be presented later in this chapter.

2.3 VAX/VMS DISK FILE STRUCTURES

The Files-11 structure resides on a volume, which is the physical medium or the actual disk pack. The volume consists of a set of

ordered blocks, numbered physically from zero to N, where N is the size of the disk. The VAX/VMS system recognizes two disk file structures: Files-11 On-Disk Structure Level 1 (ODS-1) and Files-11 On-Disk Structure Level 2 (ODS-2). Files-11 On-Disk Structure Level 1 is used for compatibility with other Digital operating systems such as RSX and IAS, whereas the Files-11 On-Disk Structure Level 2 is used by VMS. Some special files, known as reserved files, are needed to completely desribe either of these two structures. The discussion that follows concerns only the reserved files related to Level 2 (which is a superset of Level 1). Each of the files discussed may be found in the master file directory (MFD) of the disk (Digital, "Guide to VAX/VMS Disk and Magnetic Tape Operations).

2.3.1 The Index File

The index file is the most important file of the Files-11 structure. Inside the index file, there are many pieces of information that describe the disk and all of its files. The volume's bootstrap block is virtual block number one of the index file. If the volume is a system device, the bootstrap block contains a program that loads the operating system into memory. The second virtual block of the index file is commonly referred to as the home block. The home block contains device specific information such as the volume name, maximum number of files, and volume ownership information. Files-11 volumes also provide a backup home block as part of the index file in case the original home block is destroyed. Finally, the majority of the index file is made up of file headers. Each file on the volume has a file header, which details information such as the file owner, file

protection, and file location on the disk. The file header points to all the information needed for gaining access to the actual file located on the disk. The index file is found in the MFD and is named INDEXF.SYS;1.

2.3.2 The Storage Bit Map File

The storage bit map file controls the available space on the disk. It contains a control block as logical block number one of the file. This control block is used to summarize the Files-11 allocation data on the disk. Such information includes the device cluster size, number of free blocks, and maximum number of blocks. The remaining blocks of the file are used for the bit map. Basically, each bit in the file denotes whether a corresponding block on the volume is available. The bit map file is found in the MFD and is named BITMAP.SYS;1.

2.3.3 The Bad Block File

The bad block file is a special file used to hold all known defective blocks on the volume. Once the operating system detects that a block may be bad, it is written to this file. However, if the suspected block is part of a valid file, it will only be revectored when the file is deleted. The bad block file is found in the MFD and is named BADBLK.SYS;1.

2.3.4 The Master File Directory

The master file directory (MFD) is the root of the volume's directory structure. It contains all of the reserved files for the

volume as well as the root of all user directory trees. The master file directory is listed in the MFD as 000000.DIR;1.

2.3.5 The Core Image File

The core image file is listed in the MFD as CORIMG.SYS;1. It is not (yet) supported by VMS.

2.3.6 The Volume Set List File

A volume set, in the VMS world, is the ability of the operating system to look at a collection of disks as one large, consolidated disk. The ability of the operating system to view several physical disks as one larger disk has several distinct advantages. When a disk is the first volume of a multi-volume set, the volume set list file is used. This file contains the name of all the disks in the set and the set's logical name. The volume set list file is found in the MFD and is named VOLSET.SYS;1.

2.3.7 The Continuation File

The continuation file is also used when manipulating volume sets. The purpose of the continuation file is to aid file lookup when a file crosses from one volume to another in a volume set. The continuation file is found in the MFD and is named CONTIN.SYS;1.

2.3.8 The Backup Log File

The backup log file is listed in the MFD as BACKUP.SYS;1. It is not (yet) supported by VMS.

2.3.9 The Pending Bad Block File

The pending bad block log file contains a list of suspected bad blocks on the volume that are not a part of the bad block file. The pending bad block file is found in the MFD and is named BADLOG.SYS;1.

2.4 DISK STORAGE SYSTEM ARCHITECTURE

Thus far, the discussion of disk system management has been strictly from a software viewpoint. However, it is important to understand the hardware aspects related to disk system management since the hardware is a major limitation in disk system performance. According to Digital, seek time and rotational delay use 78 percent of the total transfer time on a VAX 11/780 to complete a disk transfer of four to eight blocks (see the figure 2-2) (Fisher, "Advanced System Management Techniques"). To truly understand how to optimize a disk, the disk system manager must not only understand the on-disk information structure, but also understand the disk input/output (I/O) mechanism. The following sections are an attempt to establish this understanding.

TRANSFER COMPONENT	COST	MOST IMPACTED BY:
I/O REQUEST SETUP	4 %	HOST CPU SPEED
CONTROLLER LATENCY	2 %	CONTROLLER TYPE (HSC, UDA, ETC)
SEEK TIME	58 %	FIRMWARE IN CONTROLLER
ROTATIONAL LATENCY	20 %	ROTATIONAL SPEED OF DISK
TRANSFER TIME	12 %	DENSITY, FORMAT OF DISK
I/O POST PROCESSING	4 %	HOST CPU SPEED

Figure 2-2. Disk Transfer Time Percentages

2.4.1 Introduction To Disk Storage Architectures

The most advanced disk storage system implemented by Digital is called Digital Storage Architecture (DSA) (Sun, "Disk I/O: Part I"). In DSA, an intelligent controller is used to interconnect the host internal bus and the disk drive. This architecture does more than add another smart controller to offload drive control from the CPU. In the DSA specifications, the interface between the host computer, the disk controller, and the disk drive are very clearly defined. Furthermore, message types and formats allowed, are all explicitly defined in the DSA specifications. A design of this type has many distinct advantages, but the biggest advantage is any layer (device driver, controller, or the disk itself) can be reprogrammed or replaced altogether, so long as the interface between the adjacent layers stays the same. Other types of disk storage architectures implemented by Digital and various third-party vendors are based on UNIBUS and MASSBUS disk storage subsystems. However, these other architectures are

quickly becoming obsolete (they are over ten years old), so the discussion here will focus on the functions performed by DSA.

2.4.2 Seek And Rotational Optimization

Several techniques are employed in the seek and rotational optimization algorithms. The seek ordering technique is used to minimize the disk head movement by ordering all I/O requests so that the closest seek will be satisfied first. Also, for the case where a single controller controls multiple drives, overlapping seeks are allowed. This technique enables one drive to perform a data transfer while the other drives are seeking or are idle. These techniques are designed to reduce disk latency and improve throughput.

2.4.3 Data Error Checking And Correction

Most disks perform some type of automatic data error checking and correction. These include ECC correction, use of checksums, and other transparent features (Fleischman, "ODS-2 Disk Optimization"). Automatic error correction dictates that the disk, its controller, or its software device-driver, will go out of its way to recover from disk errors without the user's immediate knowledge. This is a feature that is not without its cost. Error correction takes time and can, in some instances, cause a mysterious loss of performance. This is due to the fact that seeks requiring error recovery, typically take ten times as long as an error-free seek. DSA provides a means for all error checking and correction at each of its layers.

2.4.4 Logical To Physical Data Mapping

Perhaps the single most important function performed under DSA is the offloading of the logical to physical data mapping to the controller. When the host computer requests a data block from secondary storage, only the logical block number (LBN) is given to the controller. The controller is then responsible for determining the physical block location; that is, cylinder, track, and sector, on the drive. With this feature, the host disk driver is freed from knowing the details of the disk geometry and is able to control devices with varing characteristics such as capacity and speed.

2.5 ODS-2 DISK OPTIMIZATION

Now that the basic concepts related to disk management have been presented, it is time to direct this discussion to actual disk management. To begin with, please note that tuning is a very tedious and time consuming process. A disk system manager cannot expect to monitor disk activity for a single day and then be able to optimize the disk to its peak performance. Performance tuning is an objective, repetative cycle that is best undertaken in a systematical approach. A tuning cycle should minimally consist of data gathering and data analysis, which can then be followed by the needed tuning action. When tuning does occur, only one or two parameters should be changed at any given time. Once these changes have been made, time should be allotted to verify the changes were made for the better. Significant performance improvement should be expected only when a system is badly out of tune. On an average system in an average environment, a ten

percent improvement should be considered substantial (Digital, "Guide to VAX/VMS Disk and Magnetic Tape Operations).

2.5.1 The Tuning Cycle

VMS provides adequate tools for gathering data to be used during performance analysis. Other products, available from Digital or other third-parties, often provide a greater assistance in gathering data because they may automate some processes, produce better reports, provide charts and archive capabilities, or simply integrate many important aspects of an assortment of utilities into one package. The discussion that follows has narrowed its scope to include only those tools available with standard VMS. (For a complete discussion on tuning VMS, see Digital, "Guide to VAX/VMS Performance Management".)

When beginning a tuning cycle, a disk system manager must be prepared to tackle three major disk performance issues.

1. Volume Fragmentation

2. Adequate Free Space

3. Adequate Memory for Disk Caches

Volume fragmentation makes every VAX/VMS system slow down. It causes VMS to appear very slow even under a light user load. With every fragmented file comes slower response and with slower response, the number of I/O operations queued to a volume begin to increase. The term volume fragmentation has two meanings. The first meaning is used to describe the situation where an individual file is not contiguous on

disk. In this case, the file is broken up into pieces and scattered around the disk. This occurs because VMS always (by default) tries to allocate space for a file as close to the beginning of the disk as possible. VMS will do this even if there is enough contiguous free space elsewhere on the disk. As the number of extents in a file increases, the response time required to read the file gets slower and slower.

The second meaning of volume fragmentation is used to describe a condition in which all the free space consists of a few clusters scattered in many locations around the disk. When this occurs, it may be impossible to defragment a file unless there is a set of contiguous free blocks greater than the size of the file. Free space fragmentation is as much a problem as file fragmentation.

There are three basic tools available on VMS to monitor fragmentation and its effect on performance. The first tool is the MONITOR FCP utility. This tool displays the "Window Turn Rate" for files. The window turn rate is an indication of file fragmentation overhead being imposed on VMS (Fleischman, "ODS-2 Disk Optimization). Behind the scences, VMS creates window control blocks to map all extents of an opened file. When a specific extent of the file is needed, the window control block is searched for the file extent address. If it is not currently resident in the window control block, a physical read must occur to get the requested data into memory. Also, while examining this display, notice the "CPU Tick Rate". This indicates how much CPU time is being used by VMS to handle file

requests. Obviously, excessive CPU time indicates that the operating system is being forced into handling badly fragmented files.

A second tool available from VMS for monitoring file fragmentation is the MONITOR FILE_SYSTEM_CACHE utility. This utility displays the various cache buffer hit rates for the system. VMS uses cache buffers to speed up disk file accesses by reading multiple blocks on each physical read. An important parameter to observe on this display is the file header hit percentage (FILE_HDR). If this value is large, then either a high window turn rate or a very high file open rate could be causing disk performance problems.

The third tool available from VMS for monitoring file fragmentation is the DUMP/HEADER utility. This routine operates on a specified file and can be used to list the number of retrieval pointers used to map a file. An individual retrieval pointer is required for each extent. If many retrieval pointers are present on a highly accessed file, use the COPY/CONTIGUOUS command to create a new version of the file as one contiguous extent.

By using the above tools, the disk system manager can acquire performance data related to file fragmentation. If file fragmentation is determined to be a problem, VMS provides a means for defragmenting the files. The BACKUP utility is the traditional method, but this is a long process. First, a full backup of the entire disk must be performed. Second, the disk must be reinitialized and finally, the full backup is restored to the disk. Since the disk is blank after it is initialized, each file restored from the full backup will be contiguous on the clean disk. This method does suffice in that it does

defragment all of the files. However, this method has several disadvantages. These include requiring an operator to be present during the entire backup operation and having to isolate the disk from any useful work. Furthermore, there is always the risk of total data loss if the backup is not valid.

Many third party software vendors have developed products that perform disk defragmentation on a running system. These products operate in a way that is transparent to the users of the system, thereby eliminating some of the disadvantages of traditional backup and restore operations. These software packages are very useful and are fast becoming a necessity at all VAX sites. However, Digital has announced plans to provide a utility for defragmenting files. Regretfully, no release date on this utility has been given.

A second major hurdle associated with disk system management is related to the availability of free space on the volume. Disks should be reorganized often and at least ten percent of the volume's storage should be kept free. This reduces the rate at which free space is recycled for new file allocations. Disk fragmentation levels escalate rapidly when the free space is recycled frequently. If there are many files that are not accessed frequently, archive them to tape or removable disks. Failing that, move these files to the outer regions of the disk. This will insure that the disk's read and write heads will not have to skip over them while searching for frequently accessed data.

The best tool that VMS provides for free space management is the SHOW DEVICE D command. This command displays all of the disks available on the system and the number of free blocks on each device. The use of this command should be a standard part of routine disk management because when free space suddenly disappears, performance problems can result.

Yet another major performance issue is the use of system caches and their effectiveness. There are many system caches that relate to disk performance including the bit map cache, the file header cache, and the directory header cache. The bit map cache controls how many blocks of the allocation file BITMAP.SYS are loaded into memory. The more blocks loaded into memory, the less the number of accesses required to disk when allocating a new file or extending an old one. The file header cache controls how many file headers are loaded into memory from the file INDEXF.SYS. When file headers are loaded into memory, file lookups are quicker. The directory header cache controls how many blocks of directory files are loaded into memory. Similiar to file headers, the more directory headers that are found in memory, the shorter the time required for a full file lookup. All in all, for system caches to be effective, they must be monitored and their significance must be understood.

To examine the caching values and their hit rates, the MONITOR FILE_SYSTEM_CACHE can again be employed as well as the use of the ANALYZE/SYSTEM command. When the monitoring of these displays indicates a need for caching adjustments, the system generation utility (SYSGEN) must be invoked (Digital, "Guide to VAX/VMS Performance

Management"). Remember that by increasing the size of disk caches, the amount of free memory available on the system will decrease. A good disk system manager will be able to tune not only the disk subsystem of VMS, but will respect the ramifications that disk tuning may have on the rest of VMS.

All in all, if file fragmentation can be minimized and a healthy reserve of free space can be maintained, disk management may require very little involvement. The following paragraph details some disk management rules that are widely accepted in the VMS community as being standard rules of practice (Fisher, "Advanced System Management Techniques").

1. Thirty to thirty-five I/O operations per second is the maximum rate for any disk. This also indicates an overworked disk.

2. Fifteen or more I/O operations per second per disk (fifty percent of the maximum) usually are cause for at least a closer investigation.

3. Average transfer sizes for a non-system disk are usually four to sixteen blocks.

4. Sixty or more I/O operations per second over all disks are typical causes for concern. Do not lose track of what applications are being run.

5. Window turns are caused by fragmentation. An increase in the average number of window turns indicates fragmentation is destroying disk performance.

6. A good hit rate for disk system caches is seventy percent or better.

2.6 SUMMARY

This chapter began by discussing the history of file storage media. It was shown how storage media has progressed from filing cabinets, to card boxes, to magnetic tapes, and finally, to the disk media technology used today. A brief discussion of the concepts related to VMS disk usage was then presented. This forms a basic foundation for the terminology used in later chapters. A quick look at the underlying hardware was also briefly mentioned. Finally, the discussion turned to actual disk system management and the major hurdles that must be overcome to manage the disk I/O subsystem on VMS. Some basic guidelines and standard rules for specific devices were presented.

CHAPTER 3

DISKMGR USER'S GUIDE

The disk manager (DISKMGR) utility has many advantages over the current VMS system management design. This chapter provides a look at the many functions available and how these functions do indeed make disk management easier. Also, this chapter will present many interesting insights into the utility and it should provide the reader with sufficient reference material to fully operate and understand DISKMGR.

3.1 WHY USE DISKMGR?

The DISKMGR utility's goal is to monitor disk performance without encountering the stumbling blocks found under the current VMS system management architecture. That is, DISKMGR attempts to eliminate many of the problems discussed in chapter I. To accomplish this goal, DISKMGR provides one utility that serves as a single focal point for disk system management. Several advantages can be realized when executing the DISKMGR utility and are worthy of discussion.

One of the biggest advantages of the DISKMGR utility concerns the accessibility of information. Presently, VMS forces its disk system managers to know several utilities in order to monitor the system's performance. This restriction dictates that the disk system manager fully understand, and be confident in operating, many existing VMS utilities. Many of these utilities have different user interfaces which serves to confuse matters further. DISKMGR provides a single source utility, with a consistent user interface, designed to retrieve all pertinent information needed for disk system management. DISKMGR does not force a user to invoke several utilities to gather the needed data, rather it integrates the important aspects from an assortment of existing utilities onto one display. Therefore, information accessiblility is seen as an advantage for using DISKMGR over traditional VMS techniques.

Another advantage of the DISKMGR utility is its ability to educate new disk system managers. Through the use of various help modules (discussed in detail later) and the use of the execute command function key (also discussed later), the user is provided with a useful training tool. The help modules provide the user with an understanding of the definition of the data being displayed, as well as an understanding of how this data relates to disk management. Meanwhile, the execute command function key serves as an introduction to other VMS commands that are useful for proper disk management. All of this implies that any user can derive benefits from this utility without having to be a VMS wizard.

As seen from the above discussion, DISKMGR is a utility that allows non-expert users to make better informed disk management decisions in less time. In many instances, DISKMGR may be the only tool required for disk management. If a utility can provide its user with a summary of pertinent disk performance statistics, and a means to understand these statistics, then that utility may become a tool capable of total disk management. DISKMGR hopes to achieve this goal and will settle for nothing less. The following sections of this chapter discuss many of the operation aspects involved with the setup and execution of DISKMGR.

3.2 DISKMGR DIRECTORY STRUCTURE

The DISKMGR utility uses a series of logical names to define various directories that contain pieces of the DISKMGR package. In this section, each of the required logical names, and the files they point to, are discussed.

3.2.1 MS_CODE

This logical name should point to a directory that contains all of the FORTRAN and MACRO source code used to build the DISKMGR utility. These files are compiled and assembled during the building of DISKMGR. All of the object files created are placed into an object library which also resides in this directory.

3.2.2 MS_COM

This logical name should point to a directory that contains a series of command files useful in the building of the DISKMGR utility. Command files have been created to build each portion of the DISKMGR utility. For example, a file named CREATE_OBJLIB.COM can be used to create the object library found in the MS_CODE directory.

3.2.3 MS_INC

This logical name should point to a directory that contains all of the include files used by the modules that form DISKMGR. Include files are an important part of the DISKMGR design because they allow the definition of data structures and parameters to occur in one isolated file. All modules that reference include files are assured that they are each using the same definitions to do their work.

3.2.4 MS_EXE

This logical name should point to a directory that contains the DISKMGR executable image. Also, this directory contains the DISKMGR.CLD file that is used to define the DISKMGR command (see the corresponding section in this chapter).

3.2.5 MS_DOC

This logical name should point to a directory that contains all of the documentation associated with the DISKMGR utility. For example, this manual is found in the MS_DOC directory.

3.2.6 MS_HELP

This logical name should point to a directory that contains the help library used by the DISKMGR utility. This help library must exist if the help capabilities are to be used. The help text file and its corresponding library are both found in this directory.

3.2.7 MS_MESS

This logical name should point to a directory that contains the DISKMGR error message source file. This file is compiled by the VMS Message utility to create a set of message codes that are used by the DISKMGR utility. (For a complete discussion of all possible error messages, see the corresponding section in this chapter.)

3.2.8 MS_TOOLS

This logical name should point to a directory that contains a set of useful VMS command procedures. These procedures were used during the development of the DISKMGR package and are provided only for the interest of the reader.

3.3 USING THE HELP LIBRARY

The DISKMGR utility comes complete with a help library that provides interactive help during run-time to the user. This library is accessed by the DISKMGR utility during its execution in order to retrieve help on the meaning and definition of the data being displayed. Therefore, the user must minimally insure that the help

library, DISKMGR.HLB, does exist in the directory MS_HELP. This can be done by executing the command procedure CREATE_HLPLIB.COM that is found in the MS_COM directory. Once the DISKMGR help library has been created, help will then be available to the interactive user of the utility.

In addition to the help library being accessible from inside the DISKMGR utility, the help library can also be accessed from the VMS command level by the VMS command HELP. To notify the VMS HELP command to access the DISKMGR help library, the following logical name definition should be made:

```
$ DEFINE HLP$LIBRARY MS_HELP:DISKMGR
```

This definition will then allow the VMS HELP command to locate the DISKMGR help library and make it accessible to the user. There are several other methods that can be employed to access the library from the VMS command level. (For a description of these other methods, refer to Digital, "VAX/VMS Librarian Reference Manual".) Also, the curious user may find a method of adding the DISKMGR help library to the system-wide help library. Although this may be done, it is beyond the requirement for operating the DISKMGR utility.

3.4 EXECUTING DISKMGR

This section describes the DISKMGR operating environment and its command syntax. Many examples of the DISKMGR utility, with its different command qualifiers, are also presented.

3.4.1 DISKMGR Command Definition

The DISKMGR utility has been developed to take advantage of the command line definition features available under VMS. These features allow a user to define their own commands. In order to use these features, a command line definition file must be created. Figure 3-1 shows the command line definition file used to define the DISKMGR command.

```
DEFINE VERB DISKMGR
     IMAGE "MS EXE:DISKMGR.EXE"
     QUALIFIER DEVICES, VALUE(LIST)
     QUALIFIER FULL
     QUALIFIER EXCLUDE,  VALUE(LIST)
     QUALIFIER ALL
     QUALIFIER UPDATE, VALUE(TYPE=$NUMBER)
```

Figure 3-1. DISKMGR Command Definition File

With VMS, all commands available to a user are defined in the user's process command table. This table is searched each time a command is entered. If the command does not exist in the command table, an error will result. In order to add the DISKMGR command to the user's process command table, the VMS command

```
$ SET COMMAND MS_EXE:DISKMGR
```

should be used. The DISKMGR command will then be known to VMS and will be treated as any other VMS command.

However, by defining DISKMGR in the above fashion, the command is only defined in the user's process command table. This table exists only for the duration of the process and is recreated each time the user logs onto the system. Therefore, to make the command available each time a user logs in, place the SET COMMAND command in the default login command file of the user.

Alternatively, the DISKMGR command can be added to the VMS command table that is known by every process on the system. To add a command to the VMS system-wide command table, use the SET COMMAND command with the /TABLE qualifier and also specify the table SYS$LIBRARY:DCLTABLES.EXE. The following example adds the DISKMGR command to the system-wide command table.

```
$ SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES-
_$ /OUTPUT=SYS$LIBRARY:DCLTABLES MS_EXE:DISKMGR
```

By defining the command with this method, it becomes available to all users the next time they log in. However, when a VMS upgrade is performed, the DISKMGR command will be lost and will need to be added to the new version of the VMS command table. Therefore, it is recommended that the command only be added to the process command table. (For other restrictions and manipulations that may be performed on a user's command table, the interested reader should consult Digital, "VAX/VMS Command Definiton Utility Reference Manual".)

3.4.2 DISKMGR Command Syntax

The DISKMGR command allows the use of five qualifiers on the command line. These qualifiers are /ALL, /DEVICES, /EXCLUDE, /UPDATE, and /FULL. Each of these qualifiers is provided in order to increase the functionality and flexibility of the DISKMGR utility. The use of these qualifiers are described in the following paragraphs.

3.4.2.1 /ALL

Use the /ALL qualifier to flag the DISKMGR utility to monitor all known disk drives. This qualifier is the default if neither of the qualifiers /ALL or /DEVICES are specified. Use of this qualifier requires the user be able to create a subprocess.

3.4.2.2 /DEVICES

Use the /DEVICES qualifier to force the DISKMGR utility to monitor only a list of specified devices. The specified device list can consist of any disk on the system. The devices may be specified by using either the physical device name or a pre-assigned logical name, provided each specified device is a disk. If the /ALL qualifier is specified, this qualifier will be ignored.

3.4.2.3 /EXCLUDE

Use the /EXCLUDE qualifier to list devices that are not to be monitored by DISKMGR. The excluded device list can consist of any disk

on the system. The excluded devices may be specified by using either the physical device name or a pre-assigned logical name.

3.4.2.4 /UPDATE

Use the /UPDATE qualifier to specify the DISKMGR display update rate. By default, DISKMGR attempts to update the display every five seconds. However, this qualifier allows the user to change this default value. The specified update rate must be an integer value greater than zero and is interpreted as the number of seconds between display updates.

3.4.2.5 /FULL

Use the /FULL qualifier to flag the DISKMGR utility to set form number two as its initial display. By default, the initial display consists of all specified devices on form number one. Use of this qualifier changes the initial display to form number two using the first device specified. A complete description of the forms available with DISKMGR is presented in another section of this chapter.

3.4.3 DISKMGR Command Examples

Different combinations of the various qualifiers will result in different actions being taken by DISKMGR. Figures 3-2 through 3-6 provide some example commands and the initial displays generated by each command.

SVT/UTSI/RGD DISK MANAGER
9-APR-1988 20:10:48.70

DEVICE NAME	VOLUME LABEL	ERROR COUNT	MAX # BLOCKS	FREE # BLOCKS	PERCENT USED	TRANS COUNT	MOUNT COUNT
*DRA3:	PAYROLL	0	1008000	183387	81.8	2	1
DUA0:	SYSTEM	0	891072	109191	87.7	124	1
DUA1:	ENGINEERING	0	891072	35862	96.0	1	1
DUA2:	ACCOUNTING	0	891072	55866	93.7	2	1
DUA3:	DRIVE3	0	891072	57186	93.6	1	1
	TOTALS:	0	4572288	441492	90.3	130	5

Figure 3-2. DISKMGR

This command demonstrates all of the defaults associated with DISKMGR. That is, all disks on the system are to be monitored, the initial display will be form number one, and the update rate is five seconds.

SVT/UTSL/RGD DISK MANAGER
9-APR-1988 20:12:22.69

DEVICE NAME	VOLUME LABEL	ERROR COUNT	MAX # BLOCKS	FREE # BLOCKS	PERCENT USED	TRANS COUNT	MOUNT COUNT
*DRA3:	PAYROLL	0	1008000	183387	81.8	2	1
DUA0:	SYSTEM	0	891072	109191	87.7	124	1
DUA3:	DRIVE3	0	891072	57186	93.6	1	1
	TOTALS:	0	2790144	349764	87.5	127	3

Figure 3-3. DISKMGR/EXCLUDE=(DSK$DISK1,DSK$DISK2)

This command is almost the same as the one shown in the previous example. The only difference is that two disks have been excluded from the display. Otherwise, both commands have the same initial display and update rate.

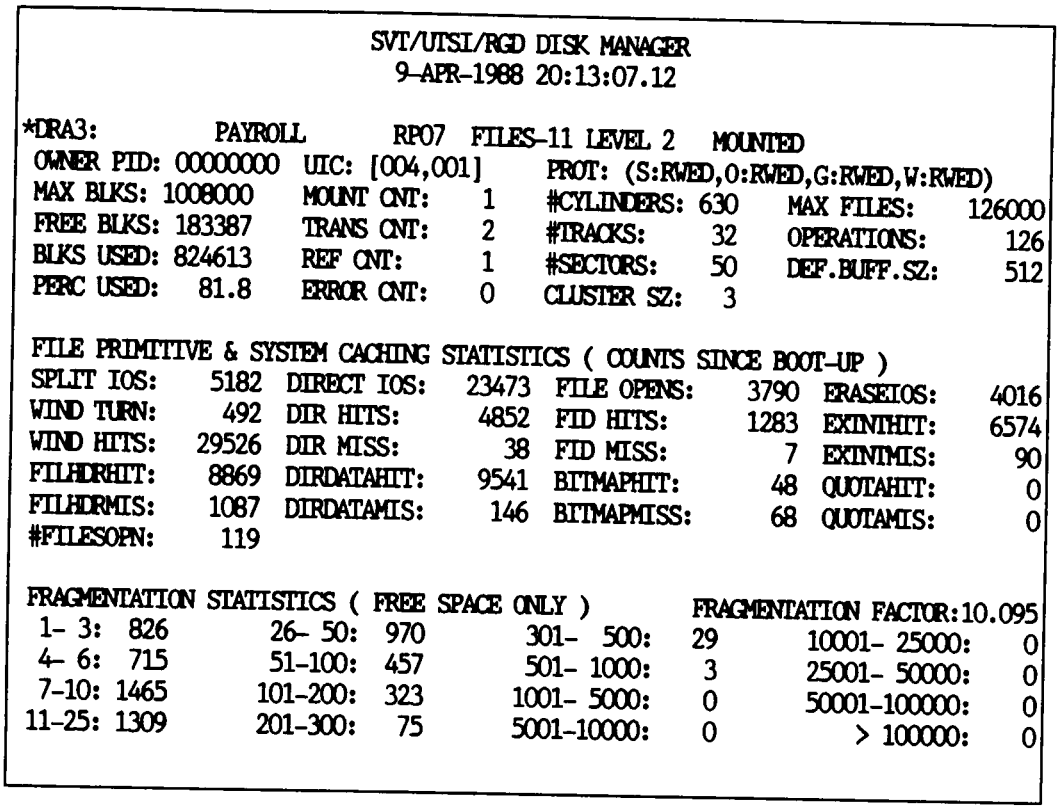


Figure 3-4. DISKMGR/UPDATE=10/FULL

This command will monitor all disks on the system. The initial display will be form number two since the /FULL qualifier was specified. Also, the display update rate will be ten seconds instead of the usual five.

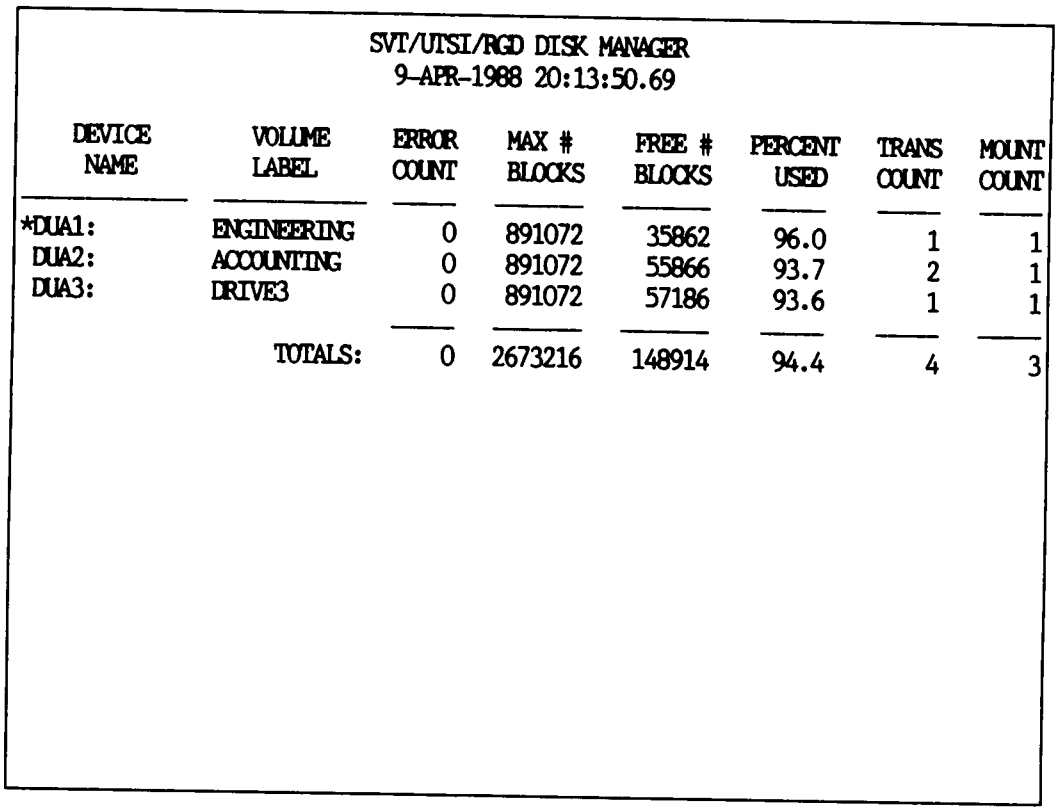


Figure 3-5. DISKMGR/DEV=(DSK$DISK1,DSK$DISK2,DSK$DISK3)

This command demonstrates the /DEVICES qualifier. As seen in this example, only the specified disks are being monitored. The initial display is still form number one and the display update rate is five seconds.

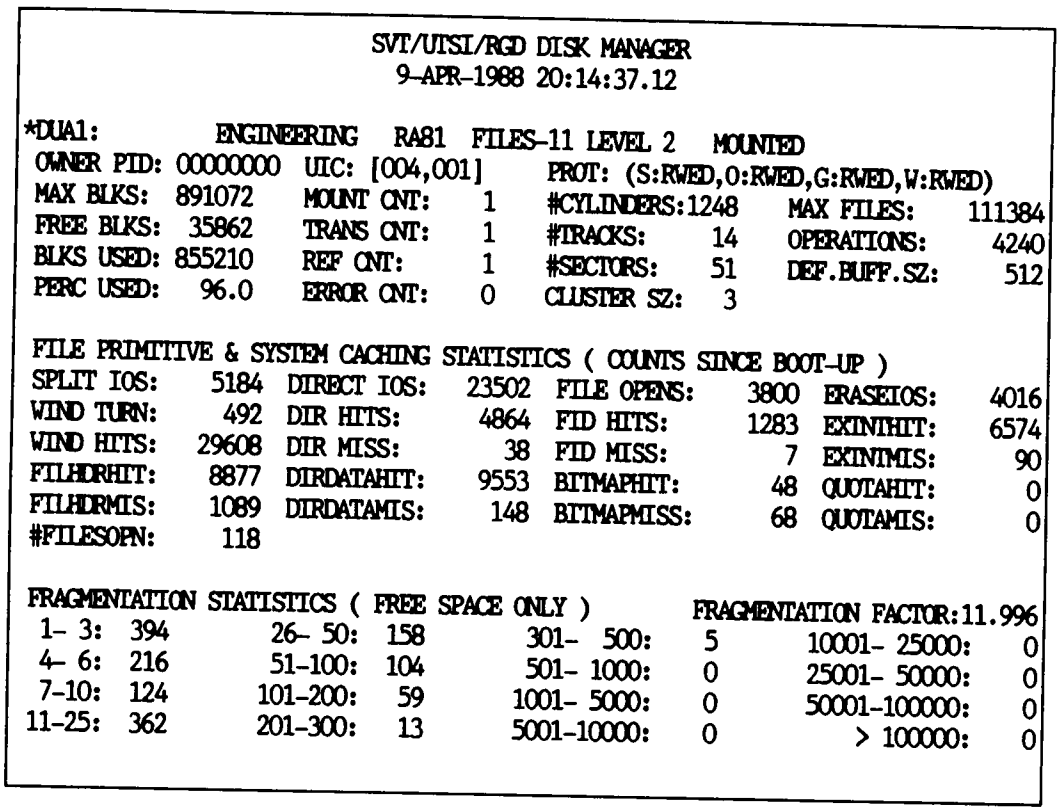


Figure 3-6. DISKMGR/FULL/DEV=(DSK$DISK1,DSK$DISK2,DSK$DISK3)

This command is almost the same as the one shown in the previous example. The only difference is that the initial display has been set to form number two. Otherwise, both commands have the list of disks and the same update rate.

3.5 DISKMGR FORMS

There are two different types of screen displays associated with the DISKMGR utility. The first display, called form number one, is a brief list of all the devices being monitored by DISKMGR. The second

display, called form number two, is a full screen listing, containing information on a particular disk, and the VMS file system. These two forms are very important to the DISKMGR utility and their meaning should be fully understood by the user. Figures 3-7 and 3-8 present a quick look at each form.

SVT/UTSL/RGD DISK MANAGER
9-APR-1988 20:10:48.70

DEVICE NAME	VOLUME LABEL	ERROR COUNT	MAX # BLOCKS	FREE # BLOCKS	PERCENT USED	TRANS COUNT	MOUNT COUNT
*DRA3:	PAYROLL	0	1008000	183387	81.8	2	1
DUA0:	SYSTEM	0	891072	109191	87.7	124	1
DUA1:	ENGINEERING	0	891072	35862	96.0	1	1
DUA2:	ACCOUNTING	0	891072	55866	93.7	2	1
DUA3:	DRIVE3	0	891072	57186	93.6	1	1
	TOTALS:	0	4572288	441492	90.3	130	5

Figure 3-7. FORM NUMBER ONE

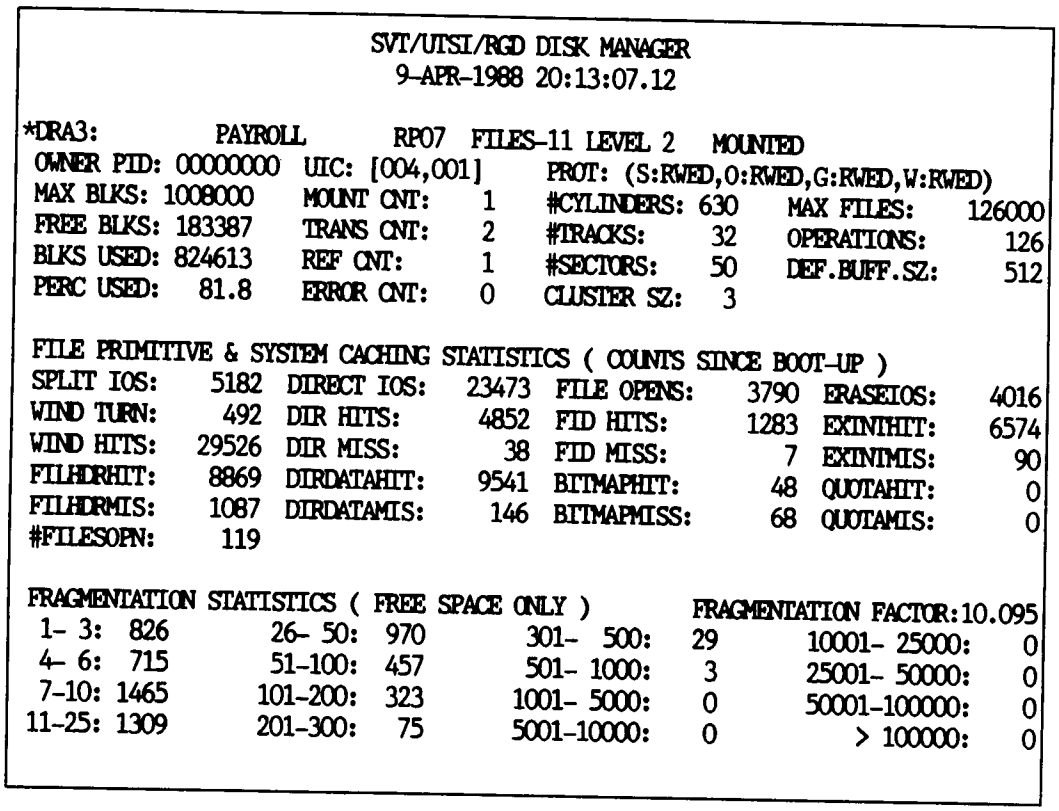


Figure 3-8. FORM NUMBER TWO

Now that each of the form layouts has been shown pictorially, a brief description of the fields on each form will be discussed.

3.5.1 FORM NUMBER ONE

Form number one consists of the standard DISKMGR header and a list of disks. For each disk being monitored, eight associated parameters are displayed. A maximum of sixteen disks can be monitored at one time. The fields displayed on form number one are discussed in the following sections.

3.5.1.1 Device Name

The device name field is used to display the name of the device being monitored. The physical device name is shown even if the user supplied a logical name.

3.5.1.2 Volume Label

The volume label field is used to display an alphanumeric name that has been encoded on the volume. This name remains on the volume until it is changed or the volume is reinitialized. That is, dismounting the volume does not effect the label.

3.5.1.3 Error Count

The error count field is used to display the number of errors on the disk. Whenever a sudden change in disk performance is noticed, the error count should be checked. A faulty disk can cause slow disk performance even on a disk with little or no activity.

3.5.1.4 Max # Blocks

The maximum number of blocks field is used to display the maximum number of blocks present on the disk. The maximum number of blocks on a disk is the total number of free and allocated blocks. This number is device dependent and cannot be changed by the user.

3.5.1.5 Free # Blocks

The free number of blocks field is used to display the number of blocks available on the disk. In general, the disk system manager should attempt to keep at least ten percent of all disk space free. By keeping ten percent of the total space free, the time period between volume restructures can be extended.

3.5.1.6 Percent Used

The percent used field is used to display the percentage of blocks in use on the disk. This percentage is calculated by dividing the number of blocks used by the maximum number on the disk.

3.5.1.7 Trans Count

The transaction count field is used to display the transaction count for the volume. This is the number of open files on the volume. The transaction count is incremented every time a file access block (FAB) is created and is decremented for each FAB deletion. It is sometimes useful to contrast the transaction count with the reference count. In general, the transaction count is greater than the reference count.

3.5.1.8 Mount Count

The mount count field is used to display the number of nodes that currently have the volume mounted. In a single CPU environment, this

value is never greater than one. However, in a VAXcluster environment, the value may be greater than one.

3.5.2 FORM NUMBER TWO

Form number two consists of the standard DISKMGR header and a set of seventy-six fields. These seventy-six fields are divided into three subsets; disk fields, file system fields, and disk fragmentation fields. Each of the individual disk fields and file system fields are discussed in the following sections. However, the disk fragmentation fields are discussed as one entry at the end of this major section.

3.5.2.1 Disk Fields

The disk fields of form number two are all of those fields that relate to a particular disk. These fields are located at the top of the display. Each field in this class will now be discussed.

3.5.2.1.1 Device Name

The device name field is used to display the name of the device being monitored. The physical device name is shown even if the user supplied a logical name.

3.5.2.1.2 Volume Label

The volume label field is used to display an alphanumeric name that has been encoded on the volume. This name remains on the volume

until it is changed or the volume is reinitialized. That is, dismounting the volume does not effect the label.

3.5.2.1.3 Media Type

The media type field is used to display the actual media type of the disk. For example, RA81 is a valid media type, as is RP06, RP07, or any other device type supporting Files-11 On-Disk Structure Level 2.

3.5.2.1.4 ACP Type Code

The ACP type code field is used to display the file structure level of the disk. Valid type are either Files-11 Level 1 or Files-11 Level 2.

3.5.2.1.5 Device Status

The device status field is used to display the current device mount status. For example, if the volume is mounted, the word "Mounted" will be displayed. If the volume is dismounted, the word "Dismounted" will be displayed.

3.5.2.1.6 Owner PID

The owner process identification field is used to identify the process that is listed as the owner of the volume. For a disk mounted with the /SYSTEM qualifier, the owner PID will always be 00000000. If the volume has been mounted to a particular process, the owner PID field will be the same as the process that has the volume mounted.

3.5.2.1.7 UIC

The owner process user identification code (UIC) field is used to identify the UIC of the volume owner. The UIC code will always be [001,001] when the disk is mounted /SYSTEM.

3.5.2.1.8 Prot

The protection field is used to display the current protection status of the volume. The protection code controls who can read, write, and delete files on the volume. Access code type execute (E) indicates create access on a volume. The protection code on a volume plays an important role in disk system security (Digital, "Guide to VAX/VMS System Security").

3.5.2.1.9 Max Blks

The maximum number of blocks field is used to display the maximum number of blocks present on the disk. The maximum number of blocks on a disk is the total number of free and allocated blocks. This number is device dependent and cannot be changed by the user.

3.5.2.1.10 Free Blks

The free number of blocks field is used to display the number of blocks available on the disk. In general, the disk system manager should attempt to keep at least ten percent of all disk space free. By keeping ten percent of the total space free, the time period between volume restructures can be extended.

3.5.2.1.11 Blks Used

The blocks used field is used to display the number of blocks allocated to the system or its users. In general, files that are seldom accessed should be archived and deleted. At the very least, these files should be moved to the outside cylinders of the disk. This will keep the read/write heads from having to scan over them when searching for a needed file. Also, heavily accessed files should be placed in the middle of the disk for optimal performance. (For infomation on deliberately placing files, see Digital, "VAX/VMS DCL Dictionary". In particular, CREATE/FDL and EDIT/FDL are useful in placing files.)

3.5.2.1.12 Perc Used

The percent used field is used to display the percentage of blocks in use on the disk. This percentage is calculated by dividing the number of blocks used by the maximum number on the disk.

3.5.2.1.13 Mount Cnt

The mount count field is used to display the number of nodes that currently have the volume mounted. In a single CPU environment, this value is never greater than one. However, in a VAXcluster environment, the value may be greater than one.

3.5.2.1.14 Trans Cnt

The transaction count field is used to display the transaction count for the volume. This is the number of open files on the volume. The transaction count is incremented every time a file access block (FAB) is created and is decremented for each FAB deletion. It is sometimes useful to contrast the transaction count with the reference count. In general, the transaction count is greater than the reference count.

3.5.2.1.15 Ref Cnt

The reference count field is used to display the reference count for the volume. This count is incremented by $ASSIGN and $ALLOC calls, and is decremented by $DASSGN and $DALLOC. The reference count is the number of channel control blocks (CCBs) pointing to the volume.

3.5.2.1.16 Error Cnt

The error count field is used to display the number of errors on the disk. Whenever a sudden change in disk performance is noticed, the error count should be checked. A faulty disk can cause slow disk performance even on a disk with little or no activity.

3.5.2.1.17 #Cylinders

The number of cylinders field is used to display the actual number of cylinders available on the volume.

3.5.2.1.18 #Tracks

The number of tracks field is used to display the actual number of tracks per cylinder available on the volume.

3.5.2.1.19 #Sectors

The number of sectors field is used to display the actual number of sectors per track available on the volume.

3.5.2.1.20 Cluster Sz

The cluster size field is used to display the cluster size of the volume. This value is the number of blocks that are allocated by a request for an additional file extent.

3.5.2.1.21 Max Files

The maximum number of files field is used to display the maximum number of files that can reside on the volume. Once this value has been set on a volume, it can only be changed by reinitializing the volume. The default value is defined to be

MAX FILES = MAX BLOCKS / ((CLUSTER SIZE+1) * 2)

and is generally sufficient.

3.5.2.1.22 Operations

The operations count field is used to display the number of operations completed by the volume. This value is a running total since the time the volume was last mounted.

3.5.2.1.23 Dev Buff Sz

The device buffer size field is used to display the device buffer size in use on the volume. This is also referred to as the device blocking size.

3.5.2.2 File System Fields

The file system fields of form number two are all of those fields that relate to file system performance. These fields are not related to any specific disk, but are important for understanding the total picture of disk management. Each field in this class is discussed in the following sections.

3.5.2.2.1 Split IOs

The split I/Os field is used to display the number of split I/O operations that have occurred since the system was booted. Split I/O operations are an indication of fragmentation and are due to larger virtual I/O operations that must be broken into smaller I/O operations by the disk driver. This occurs when the larger I/O operations are not contiguous on the disk. Note that direct disk I/O rates may appear

lower due to large number of virtual I/O operations. However, the actual operations to the disk may be higher due to spliting.

3.5.2.2.2 Direct IOs

The direct I/Os field is used to display the number of direct I/O operations that have occurred since the system was booted. Direct I/O operations are generally disk operations, but not always. Also, as described under the discussion of split IOs, this value can sometimes be misleading. Therefore, use caution when evaluating this data point.

3.5.2.2.3 File Opens

The file opens field is used to display the number of file opens that have occurred since the system was booted. Applications that do excessive file opens should generally be rewritten to minimize their effect on disk performance.

3.5.2.2.4 EraseIOs

The erase I/Os field is used to display the number of erase I/O operations that have occurred since the system was booted. Erase I/O operations occur after file deletions to prevent disk scavenging (Digital, "Guide to VAX/VMS System Security).

3.5.2.2.5 Wind Turn

The window turn field is used to display the number of window turns that have occurred since the system was booted. Window turns are

required whenever a miss occurs to the window control block. When a window turn occurs, fragmentation is generally responsible.

3.5.2.2.6 Wind Hits

The window hits field is used to display the number of window hits that have occurred since the system was booted. The ratio of window hits to window turns will provide a better estimate to the degree of performance loss due to fragmentation.

3.5.2.2.7 Dir Hits

The directory cache hits field is used to display the number of directory cache hits that have occurred since the system was booted.

3.5.2.2.8 Dir Miss

The directory cache misses field is used to display the number of directory cache misses that have occurred since the system was booted.

3.5.2.2.9 Fid Hits

The file id hits field is used to display the number of file id hits that have occurred since the system was booted.

3.5.2.2.10 Fid Miss

The file id misses field is used to display the number of file id misses that have occurred since the system was booted.

3.5.2.2.11 Extnthit

The extent cache hits field is used to display the number of extent cache hits that have occurred since the system was booted.

3.5.2.2.12 Extntmis

The extent cache misses field is used to display the number of extent cache misses that have occurred since the system was booted.

3.5.2.2.13 Filhdrhit

The file header hits field is used to display the number of file header cache hits that have occurred since the system was booted.

3.5.2.2.14 Filhdrmis

The file header misses field is used to display the number of file header cache misses that have occurred since the system was booted.

3.5.2.2.15 Dirdatahit

The directory data hits field is used to display the number of directory data cache hits that have occurred since the system was booted.

3.5.2.2.16 Dirdatamis

The directory data misses field is used to display the number of directory data cache misses that have occurred since the system was booted.

3.5.2.2.17 Bitmaphit

The bit map hits field is used to display the number of bit map cache hits that have occurred since the system was booted.

3.5.2.2.18 Bitmapmiss

The bit map misses field is used to display the number of bit map cache misses that have occurred since the system was booted.

3.5.2.2.19 Quotahit

The quota hits field is used to display the number of quota cache hits that have occurred since the system was booted.

3.5.2.2.20 Quotamis

The quota misses field is used to display the number of quota cache misses that have occurred since the system was booted.

3.5.2.2.21 #Filesopn

The files open field is used to display the number of files currently opened on the system. Having a lot of open files is not

necessarily a bad situation. However, it may present a cause for concern and should dictate at least a closer look.

3.5.2.3 Disk Fragmentation Fields

The disk fragmentation fields of form number two are all of those fields that relate to the disk's free space fragmentation statistics. These fields are located at the bottom of the display.

3.5.2.3.1 Fragmentation Factor

The fragmentation factor field is used to display the fragmentation factor value associated with the current volume. This value is a measure of the degree of fragmentation on the volume. As it increases, the more influence volume fragmentation is having on disk performance. The fragmentation factor calculation is shown below.

$$\text{FRAGMENTATION FACTOR} = \frac{\text{TOTAL NUMBER OF FRAGMENTS}}{(\ \text{FREE BLOCKS} \ / \ \text{CLUSTER SIZE}\ )}$$

3.5.2.3.2 Fragmentation Statistics

The remaining fields in the disk fragmentation class of form number two, have been grouped into the fragmentation statistics category. These fields are used to denote the number of extents of varying sizes available on the disk. For example, in figure 3-8, page 40 from above, there are 826 extents of size one to three blocks on the volume. Also, notice that there is not an extent larger than 1001.

This implies that if a user needed to create a contiguous file greater than 1001 blocks, the disk would need to be restructured. The free space fragmentation statistics shown on form number two are very useful to disk managers.

3.6 DISKMGR FUNCTION KEYS

DISKMGR uses a series of predefined function keys to allow the user to interact with the utility. These function keys are all defined on the keypad and are shown in the keypad diagram layout (figure 3-9). These keys are useful for changing the appearance of the screen, retrieving help, executing field related commands, and for moving the token on the form. (The token is a special character ('*') that is used to denote the current field selected on the form.) Each of the function keys shown in the diagram will be discussed in the sections that follow.

3.6.1 Next Field

The next field function key advances the token to the next field of the form. If the token is currently in the last field of the form, the token will be repositioned to the first field of the form. On form number one, each disk display line is a field. On form number two, each data point is a field. The keypad keys number two and six, the right and down arrows, and a control I, are all defined as next field function keys.

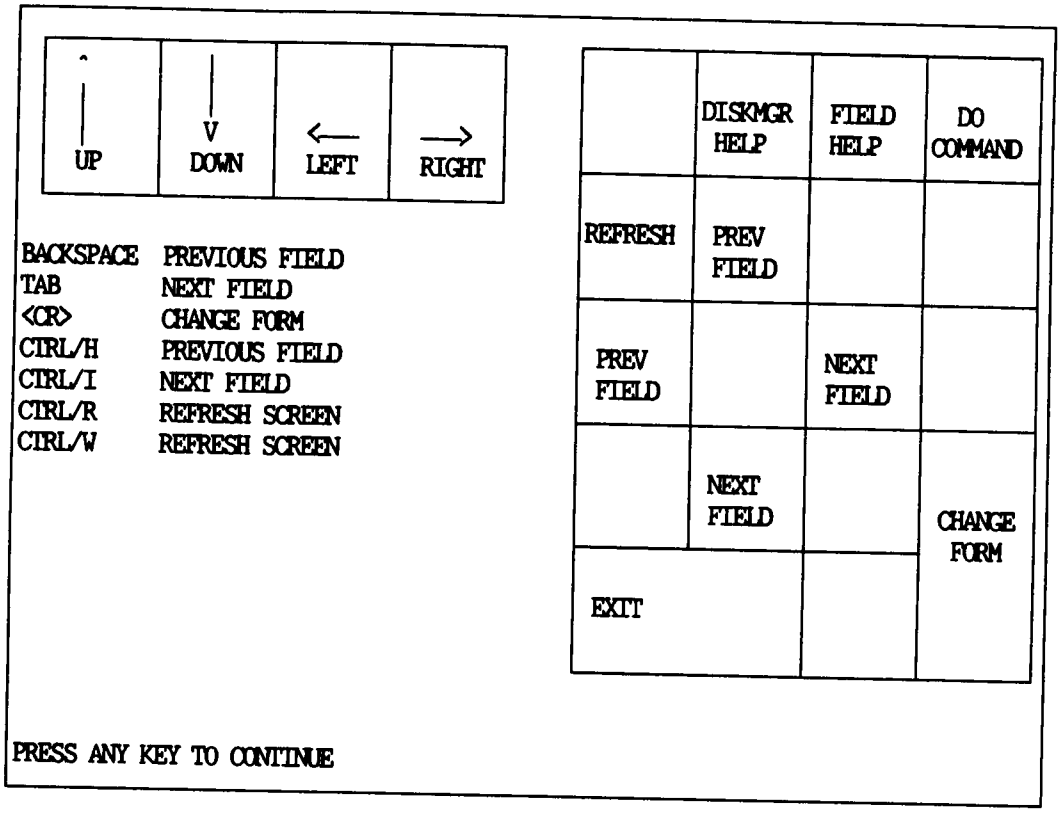


Figure 3-9. DISKMGR Keypad Layout

3.6.2 Previous Field

The previous field function key retreats the token to the previous field of the form. If the token is currently in the first field of the form, the token will be repositioned to the last field of the form. The keypad keys number four and eight, the left and up arrows, the backspace, and a control H, are all defined as previous field function keys.

3.6.3 Change Form

The change form function key removes the current form being displayed and displays the next form. When form number one is being displayed, the change form function key requests form number two to be displayed. Similarly, when form number two is being displayed, the change form function key requests form number one to be displayed. Also, when changing from form number one to form number two, the token represents the device that will be displayed on form number two. The keypad key enter, the carriage return key, the next screen key, and the previous screen key, are all defined as change form function keys.

3.6.4 Refresh Screen

The refresh screen function key redraws the current form being displayed. Since DISKMGR does not (yet) trap any broadcast messages, the display can become invalid. The refresh screen function key is provided to eliminate this problem. The keypad key seven and the control keys R and W are all defined as refresh screen function keys.

3.6.5 General Help

The general help function key displays the keypad diagram shown in figure 3-9. This display is provided such that the interactive user can quickly determine the location of a desired function key. The keypad key PF2 and the uppercase H are both defined as general help function keys.

3.6.6 Field Help

The field help function key displays the help text associated with the current field of the form. The current field is denoted by the location of the token on the form. By using the field help function key, the user is provided with an on-line help facility that explains the meaning and use of each field. The keypad key PF3 and the lowercase h are both defined as field help function keys. Figure 3-10 shows a typical screen display after activating field help.

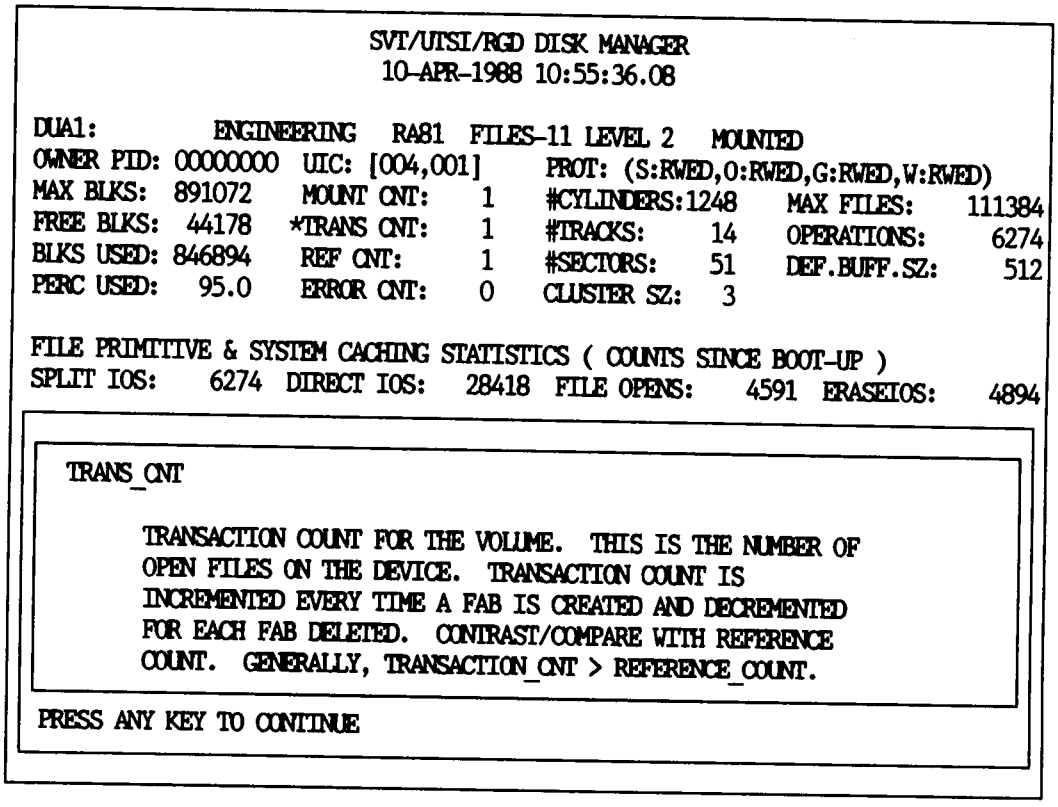


Figure 3-10. Field Help Display

3.6.7 Execute Command

The execute command function key executes a predetermined VMS command. The command that is executed is determined by the current field. By using this function key, the user is allowed to take advantage of existing VMS utilities that are appropriate for disk system management. The keypad key PF4, the D0 function key, and the letter R, are all defined as execute command function keys. Appendix C lists every field and their corresponding VMS command that is executed when selecting the execute command function key.

3.6.8 Exit DISKMGR

The exit DISKMGR function key terminates the execution of the DISKMGR utility. Usage of this function key will cause the DISKMGR utility to return control to the VMS operating system. The keypad key zero and the letter E are both defined as exit DISKMGR function keys.

3.7 ERROR HANDLING

There are several errors that can occur during DISKMGR execution. Some of these errors are caused by invalid command line values, while others may be caused by system routines, and others may occur from an input/output operation. With this in mind, the DISKMGR utility has been designed to capture many expected run-time error conditions and handle these errors in an appropriate fashion. In some instances, DISKMGR may attempt to retry an operation, while in other cases, it may be more appropriate to signal the error and then continue. Whatever the cause, DISKMGR will produce an error message on the bottom line of

the display screen. This error message will have the format shown below.

module name - error message text

In the error message, module name is the name of the module in which the error occurred, and error message text is a brief definition of the error. The following paragraphs explain all of the error message text values and their meaning.

3.7.1 Command Line Errors

"Specified device does not exist" is an error message code produced when the user has asked to monitor a device that does not exist. The user should verify that all specified devices do exist and that any logical name assignments that were used are valid.

"Unable to validate specified device" is an error message code produced when the user asked to monitor a device that could not be validated at this time. The user should verify that the requested device is available on the system.

"Specified device is not a disk" is an error message code produced when the user has specified a valid system device, but the device is not a disk. Since the DISKMGR utility will only monitor disk devices, the device class code for each specified device must be disk device.

"Maximum number of devices exceeded" is an error message code produced when the number of devices requested to be monitored exceeds the maximum number allowable by DISKMGR. Currently, the maximum number

of devices that may be monitored is limited to sixteen. This error is caused by a design restriction which assumes a twenty-four line display device where eight of these lines are reserved by DISKMGR.

3.7.2 System Service Errors

"Error spawning DCL command" is an error message code produced when an internal VMS command was attempted. This error has two probable causes. The first, and most likely cause of the error, is that the user's process has exhausted the number of subprocesses that it may create. If this is the case, the user should see the system manager for appropriate quota modifications. The second cause that should be considered is that there may not be a command line interpreter associated with the created subprocess. This can occur when DISKMGR has been created as a detached process.

"$GETDVI returned error" is an error message code produced when the VMS system service $GETDVIW returned an error code. This error is generally a severe error that will cause the DISKMGR utility to terminate. Associated with this message is another error message from the operating system.

"Error obtaining system time" is an error message code produced when the current system time could not be obtained. This error code is produced by a call to the VMS system service $GETTIM.

"Error converting time to ASCII" is an error message code produced when the VMS system time is converted into a character string format.

This error code is produced by a call to the VMS system service $ASCTIM.

"Error reading BITMAP file" is an error message code produced when DISKMGR is unable to read a device's BITMAP.SYS file. This error is most likely due to the lack of privileges on behalf of the user. To manipulate the BITMAP.SYS file, the user minimally needs read access.

3.7.3 Input/Output Errors

"Could not unpaste virtual display" is an error message code produced while trying to change forms. This error occurs when the screen management routines tried to remove a display that either did not exist or was never pasted to the pasteboard. This error will result if DISKMGR's internal I/O data structures are corrupted.

"Error saving current screen display" is an error message code produced when the user selected the execute command function key. When this function key was activated, the DISKMGR utility directed the screen management routines to save the current screen display. Again, this error will result if the internal I/O data structures are corrupted.

"Error restoring screen display" is an error message code produced when the screen management routines tried to restore a screen display. This message is usually associated with the previous message.

"Error creating pasteboard" is an error message code produced when the screen management routines were unable to allocated the specified output device. DISKMGR always directs its output to the logical name SYS$OUTPUT. The user should verify the definition of this logical name.

"Error creating virtual display" is an error message code produced when the screen management routines were unable to create a virtual display. This error will result if the internal I/O data structures are corrupted.

"Error converting value" is an error message code produced when an integer or real value is converted to character format. All values displayed by DISKMGR are of an assumed size and magnitude. If a value were to exceed these boundaries, this error message will result.

"Error writing character to display" is an error message code produced when the screen management routines encountered an error while updating the physical display.

"Error reading keystroke" is an error message code produced when the function key input routine cannot read a function key. This error is most likely caused by a user quota being exceeded.

"Error unpasting virtual display" is an error message code produced when removing a display from the pasteboard. This can occur when removing a help display or when changing forms.

3.8 DISKMGR DISCLAIMER

This public domain software is provided on an "as-is" basis. Use of DISKMGR is at user's own risk. However, this utility has been tested and a great deal of effort has been made to remove any errors. It is possible that some rare combination of factors, such as a change in the ODS-2 file structure, may cause DISKMGR to give invalid results. Therefore, this utility is furnished free of charge and the author asks to be made aware of any known errors caused by using this utility. Please feel free to notify the author with any deficiencies or recommendations.

3.9 SUMMARY

In this chapter, a complete overview of the functions available to the DISKMGR user has been presented. The means of invoking these functions, and their purpose, was explained. Also, this chapter defined the environment required for DISKMGR execution. Examples were given to guide the user through initial setup of the DISKMGR environment, an environment that includes the proper use of logical names, the use of a help library, and the definition of the DISKMGR command. This chapter also defined the meaning of every displayable field and summarized their importance in regard to disk management. Finally, a set of error codes, was presented. This chapter is structured to be read as a user's guide and it hopefully contains all of the information necessary to operate and understand the DISKMGR utility.

CHAPTER 4

PROGRAMMER'S REFERENCE MANUAL

This chapter provides a detailed look at the various subsystems involved in the DISKMGR utility. Each module of the DISKMGR utility is describe through a discussion of its algorithm. Also, this chapter summarizes the use of the VMS provided utilities and how the DISKMGR utility is used to take advantage of the powerful VMS operating system. All in all, this chapter retraces the development of the utility and provides the reader with sufficient reference material to better understand the utility as a whole.

4.1 DISKMGR

The DISKMGR utility may be decomposed into six major subsystems, each of which were developed in order to simplify the design of the overall utility. Quite a bit of time was devoted to doing a structured design of the system before any code was actually written. It was during this design process that each of the subsystems surfaced. The benefits that can be obtained from a good design are numerous and the interested reader should consult another source for more detail on how

to prepare for a structured design implementation. (For example, Page-Jones, "The Practical Guide to Structured Systems Design".)

As described in chapter III, the DISKMGR utility provides a disk system manager with a means to monitor and evaluate disk performance without encountering the stumbling blocks associated with the current VMS system management architecture. To achieve this end, DISKMGR was designed to provide its user with a clear and concise set of routines that can be an example for other programming efforts, while at the same time maintaining a design within the DISKMGR utility for later code development. In its present format, the DISKMGR utility provides aid to the disk system manager in only one aspect of the tuning cycle: data gathering. Hopefully, the design has been such that adding more capabilities, such as data analysis, will now follow with little or no redesign. This should be an important design requirement for any project, no matter the size. The DISKMGR utility does provide basic data analysis to its user through on-line field help. Although this is more than many existing utilities provide, this does not constitute data analysis in regards to the tuning cycle described in chapter II.

To briefly overview each of the subsystems, the structure chart in figure 4-1 will prove useful. As depicted in the given structure chart, the DISKMGR utility can be understood best by partitioning the system into black boxes. Each black box has its own unique function that is an essential part of the overall system, but not dependent on any other part of the system. For example, this partitioning shows that all input to the system must be handled by the Read Function Key subsystem. Unless there is a great interest in knowing how the input

from the user is handled, this subsystem can be ignored completely by someone deciphering the code. The remaining sections of this chapter are devoted to each subsystem in turn. Within each subsystem, its appropriate black box is opened up, and the details of its design are examined.

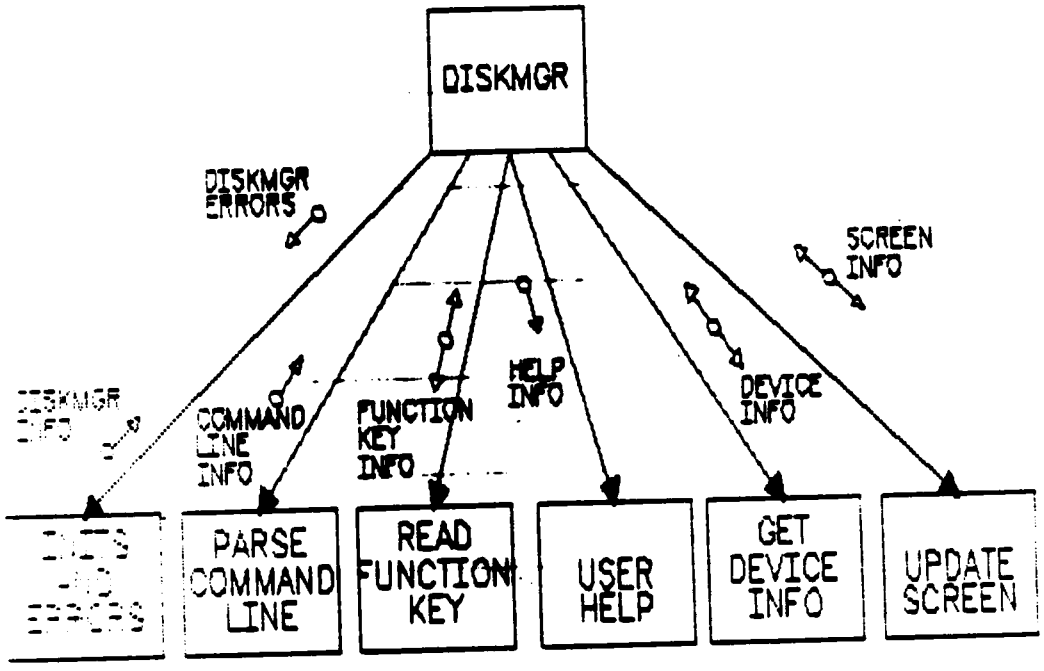


Figure 4-1. DISKMGR Top Level Structure Chart

4.2 PARSE COMMAND LINE SUBSYSTEM

The Parse Command Line subsystem performs command line input and syntax analysis for the DISKMGR utility. This section looks at this subsystem in detail.

4.2.1 Command Input And Syntax Analysis On VMS

VMS provides a means for users to create their own DCL commands. It is possible for a user to create a command complete with qualifiers and parameters such that it will look like a standard VMS command to an unsuspecting user. To create a command, a command line definition (CLD) file must be created. This file will then be applied to the user's command table by the SET COMMAND command. The DISKMGR utility implements some of the features available with the CLD utility (for a more complete reference, see Digital, "VAX/VMS Command Definition Utility Reference Manual").

4.2.2 The DISKMGR Command Line Definition

To begin the specific discussion of the DISKMGR command, the file DISKMGR.CLD is now presented.

```
DEFINE VERB DISKMGR
     IMAGE "MS_EXE:DISKMGR.EXE"
     QUALIFIER DEVICES, VALUE(LIST)
     QUALIFIER FULL
     QUALIFIER EXCLUDE,  VALUE(LIST)
     QUALIFIER ALL
     QUALIFIER UPDATE, VALUE(TYPE=$NUMBER)
```

Figure 4-2. DISKMGR Command Definition File

The CLD file starts with the command name and the image (program) that is to be executed. In the example above, DISKMGR is the command (verb) and MS_EXE:DISKMGR.EXE is the executable program (image). Five qualifiers are a part of the DISKMGR syntax. These include /DEVICES,

/FULL, /EXCLUDE, /ALL, and /UPDATE. The /DEVICES and /EXCLUDE qualifiers are defined to accept a list of arguments and the /UPDATE qualifier is defined to accept any value that can be translated to a number. The /ALL and /FULL qualifiers do not require values. The VMS command line parsing routines will not invoke the image if any portion of the specified syntax has been violated. The use of these qualifiers are detailed in chapter III of this manual. Our main concern in this subsystem is how to manipulate the command line.

Once the CLD file has been defined, the command may be added to the user's command table by the SET COMMAND command. To add the DISKMGR utility to a user's process command table, enter the following command.

```
$ SET COMMAND DISKMGR
```

Adding the command in this fashion is sufficient for the study proposed here. However, there are means in which the command can be added to the DCL table that is known system wide. The above example only adds the command to the process specific command table and will only be valid for the duration of the process. To remove the command from the user's process command table, use the /DELETE qualifier with the above command.

4.2.3 Modules Of The Subsystem

Now that the command syntax definition has been explored, it is the responsibility of the Parse Command Line subsystem to determine the

presence or absence of each command line qualifier. VMS provides command line interface (CLI) routines within the Run-Time Library that permit checking for the presence of such qualifiers. Other CLI routines return values specified by a qualifier. The following modules comprise the Parse Commmand Line subsystem and are shown in the structure chart of figure 4-3.

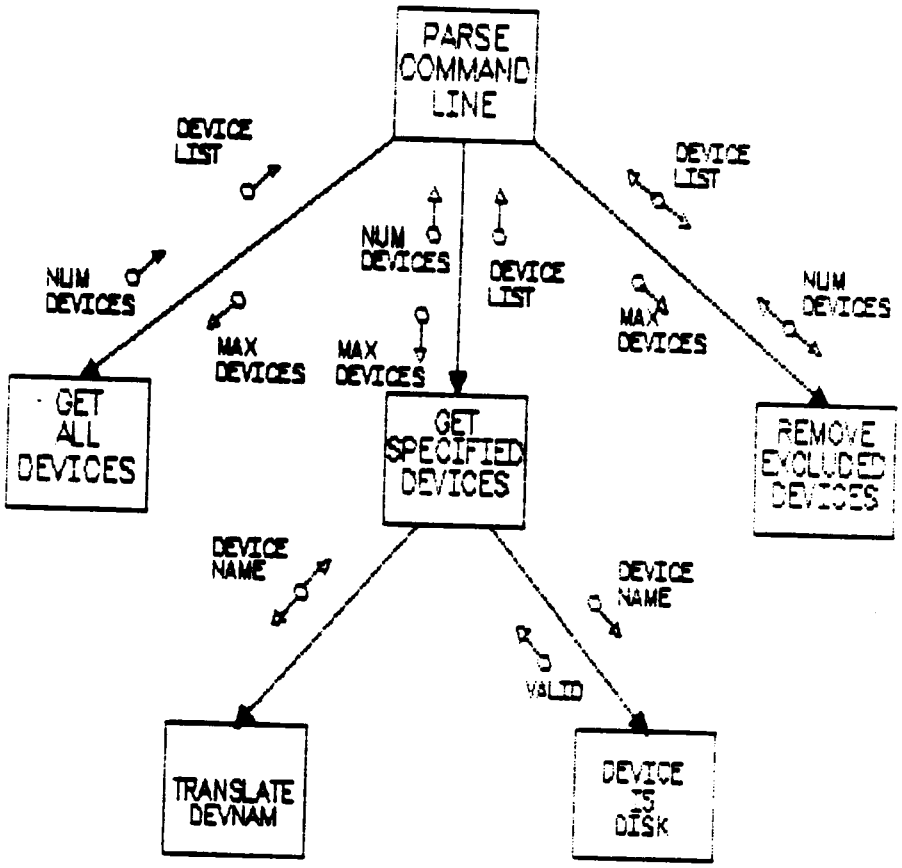


Figure 4-3. Parse Command Line Subsystem Structure Chart

4.2.3.1 PARSE_COMMAND_LINE

The PARSE_COMMAND_LINE module is the controlling module of this subsystem and is responsible for the actual run-time evaluation of the user specified command string. That is, this module will check for the presence of qualifiers, retrieve values for the specified qualifiers, provide default values for missing qualifiers, and perform command line consistency checking. To perform these operations, the Run-Time Library routines CLI$PRESENT and CLI$GET_VALUE are employed to parse the command string.

To begin with, the command line is evaluated for the presence of the /ALL qualifier. If the qualifier is present, then a call to retrieve all known disks on the system is generated. If the qualifier is not present, the command line is evaluated for the presence of the /DEVICES qualifier. If this qualifier is present, then a call to retrieve only the specified devices on the command line is generated. If neither of the /ALL or /DEVICES qualifiers are found, the module defaults to the /ALL qualifier. In either case, this portion of the module produces a (possible empty) list of devices to be monitored.

Once the evaluation of the /ALL or /DEVICES qualifier has been completed, the command line is evaluated for the presence of the /EXCLUDE qualifier. If the qualifier is present, then a call to remove all excluded devices is generated. All devices represented in the /EXCLUDE list will be removed from the set of devices determined in the first step of this algorithm. The output from this second step is a finalized device list to be monitored by DISKMGR.

Step three in the algorithm specifies the command line be evaluated for the presence of the /FULL qualifier. If this qualifier is found on the command line, then the initial form to be displayed will be form number two with device one from the device list. If the qualifier cannot be found on the command line, the initial form will be form number one and all devices will be displayed.

The last and final step in the algorithm is the determination of the update rate. This step begins by determining the presence of the /UPDATE qualifier on the command line. If the qualifier is present and a value is specified, then the specified value will be used as the display update rate. If the qualifier is present and no value has been specified, or, if the qualifier is not present, then the default update rate of five seconds will be used. (Please note, the update rate is an integer value which specifies the number of seconds between display updates.)

Upon termination of this algorithm, control is returned to the invoking module. A status code representing the success or failure of the module is returned as the function's value. The following paragraphs detail the modules described in the above algorithm.

4.2.3.2 GET_ALL_DEVICES

This module is invoked if the /ALL qualifier was used on the command line or if neither of the /ALL or /DEVICES qualifiers were used. The main purpose of this module is to supply its caller with a device list consisting of every possible disk drive currently

configured on the computer system. To accomplish this task, the following algorithm is executed.

First, a subprocess is created using the Run-Time Library routine LIB$SPAWN. This subprocess executes the VMS command "$@MS_COM:GETDEVS.COM" where MS_COM is the logical name of the directory where the file GETDEVS.COM resides. The specified VMS command procedure, and an example output file generated by the command procedure, are now shown in figures 4-4 and 4-5.

```
$ DEFINE SYS$OUTPUT MS_COM:GETDEVS.OUT
$ SHOW DEVICE D
$ PURGE MS_COM:GETDEVS.OUT
```

Figure 4-4. MS_COM:GETDEVS.COM

```
DEVICE   DEVICE    ERROR     VOLUME          FREE  TRANS MNT
 NAME    STATUS    COUNT      LABEL        BLOCKS COUNT CNT
DRA3:    MOUNTED       0  PAYROLL         183387     2   1
DUA0:    MOUNTED       0  SYSTEM          109584   124   1
DUA1:    MOUNTED       0  ENGINEERING      44178     1   1
DUA2:    MOUNTED       0  ACCOUNTING       55866     2   1
DUA3:    MOUNTED       0  DRIVE3           57186     1   1
```

Figure 4-5. MS_COM:GETDEVS.OUT

As can be seen, the above command procedure creates a file, MS_COM:GETDEVS.OUT, which contains a list of all disk devices on the system. When the command procedure exits, the subprocess is terminated, and control is returned to GET_ALL_DEVICES.

The second step in the algorithm is the processing of the file MS_COM:GETDEVS.OUT. The file is opened for formatted, sequential, readonly access, with the assumption that the file does already exist. The header lines in the file are skipped and true processing starts with line four of the file. The device name is stripped from the input line and a list of the device names is created. Proceesing of each line continues until end-of-file is detected or the maximum number of displayable devices is exceeded. In either case, upon termination of this module, a (possibly empty) list of device names is returned to the caller. Also, a status code representing the success or failure of the algorithm is provided as the function's value.

4.2.3.3 GET_SPECIFIED_DEVICES

This module is invoked if the /DEVICES qualifier was present on the command line. The main purpose of this module is to supply its caller with a device list consisting of every disk drive specified by the user. To accomplish this task, the following algorithm is employed.

By issuing repeative calls to the Run-Time Library routine CLI$GET_VALUE, all specified devices become known to the module. Once a device is known, it must pass a three-part validation test before it is entered into the device list. The first step of the validation guarantees that the specified device is a legally translatable logical name. The next step in the validation process guarantees that the specified device is not already present in the device list. This step eliminates duplicates from the command line. The third and final step

of the validation procedure is used to verify that the specified device is a disk. If a specified device fails any of these validations, it is not placed into the device list. Processing continues for every device found on the command line until all devices are processed, or until the maximum number of displayable devices has been exceeded. In either case, upon termination of this module, a (possibly empty) list of device names is made available to the caller and a status code representing the success or failure of the algorithm is returned as the function's value.

4.2.3.4 REMOVE_EXCLUDED_DEVICES

This module is invoked if the /EXCLUDE qualifier was present on the command line. The main purpose of this module is to supply its caller with an updated device list. The updated device list will not contain any devices that are present in the exclude qualifier's list. To accomplish this task, the following algorithm is incorporated.

Just as with the /DEVICES qualifier, repeative calls to CLI$GET_VALUE are used to determine every device in the exclude qualifier's list. Once a device is parsed from the command line, it must be translated to its most basic equivalence name. By comparing this equivalence name with each name in the device list, all excluded devices can be eliminated. Processing continues in this fashion for every device found on the command line and terminates when all devices have been processed or all of the devices in the device list have been eliminated. In either case, upon termination of this module, a (possibly empty) list of device names is made available to the caller

and a status code representing the success or failure of the algorithm is returned as the function's value.

4.2.3.5 Form Determination

Form determination is actually a few lines of code within the module PARSE_COMMAND_LINE. These few lines are very simplistic in their design and very little can be said to enhance their meaning. However, these lines do serve a useful function by determining the presence of the /FULL qualifier and then initializing the starting display to its appropriate value. That is, if the /FULL qualifier is specified on the command line, the starting display will be form number two with device one from the device list. If the /FULL qualifier is omitted from the command line, the starting display will default to form number one with a brief display of all devices in the device list.

4.2.3.6 Update Rate Determination

Update rate determination, like form determination, is actually a few lines of code within the module PARSE_COMMAND_LINE. This set of lines determines the presence of the /UPDATE qualifier and then determines the value specified by the /UPDATE qualifier. That is, if the /UPDATE qualifier is present, and a value has been specified along with the qualifier, then the user-supplied value will become the udpate rate. If the qualifier is omitted from the command line, or if there is not a value specified with the /UPDATE qualifier, the update rate will default to five seconds.

4.2.3.7 DEVICE_IS_DISK

This module is invoked during the validation of input devices by the GET_SPECIFIED_DEVICES module. This function is used to validate each specified device to determine if the device is an actual disk on the system. This module returns to its caller a logical boolean value signifing that the device is a disk (.TRUE.) or the device is not a disk (.FALSE.).

4.2.3.8 TRANSLATE_DEVNAM

This module is invoked by either of the two modules GET_SPECIFIED_DEVICES or REMOVE_EXCLUDE_DEVICES. The purpose of this module is to translate a given device name to its most primitive equivalence name. For example, a user might specify a device, MY_DISK_NAME, that could be a logical name for HSC000$DUA0:, the physical equivalence name. This module is responsible for providing the logical name translation to its caller. The reason equivalence names are needed is because VMS allows a user to define multiple logical names for any one physical name. If equivalence names are not used, then a command of the form would not produce the correct display if we assume WORKDISK and DISK2 are the same physical disk. (If the reader is not familiar with the use of logical names on VMS, see Digital, "Guide to Using DCL and Command Procedures". Suffice it to say that logical names are a very powerful and useful tool under VMS.)

```
$ DISKMGR/DEVICE=(DISK1,DISK2)/EXCLUDE=(WORKDISK)
```

4.3 READ FUNCTION KEY SUBSYSTEM

The Read Function Key subsystem performs user input for the DISKMGR utility. This section looks at this subsystem in detail.

4.3.1 User Input/Output With Screen Management Of VMS

The VMS operating system is a very powerful, well developed operating system. One of its greatest advantages, however, is its set of utilities, libraries, and interfaces that come bundled on top of the operating system. One such notable enhancement to VMS during the past major release has been the addition of a set of screen management routines that provide device-independent display capabilities. This set of routines is used extensively by the Update Screen subsystem to provide for user input. (See Digital, "VAX/VMS Run-Time Library Routines Reference Manual" for details on all of the SMG calls available for screen management.

There are several screen management routines available for reading user input from a terminal. However, when developing a useful utility, a very basic design decision must be made; the designer must decide in what format user input will occur. Should the user be allowed (forced) to use long, English-like commands to interact with the utility? Or perhaps, the user should be allowed to enter shorter, more cryptic commands and save a few keystrokes. Alas, neither of these ideas are viewed with much favor by VMS and its users. VMS tradition has taught its system programmers that the easiest and cleanest method of user interface is through the use of function keys. Editors, debuggers, and the like, all have a set of commands available in one or two keystrokes

that perform common everyday functions. Furthermore, a user can remember "PF2" for help, but may not remember the correct syntax to ready a domain in Datatrieve (Datatrieve is a report generation utility that requires the user to enter English-like commands and does not allow many abbreviations). Given this power, DISKMGR was designed to allow the use of function keys for acquiring user input.

4.3.2 Modules Of The Subsystem

Now that the reasoning for the use of function keys has been detailed, the discussion can now concern itself with the how, rather than the why. It is the responsibility of the Read Function Key subsystem to acquire the function key input, process the selected key, and then perform the option specified by the key. The modules created from this subsystem are shown in figure 4-6.

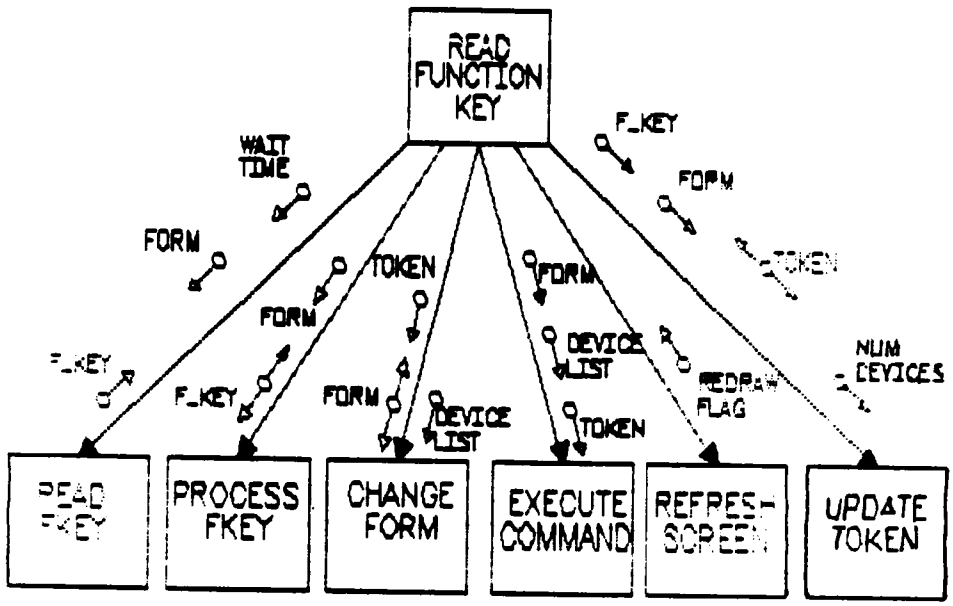


Figure 4-6. Read Function Key Subsystem Structure Chart

4.3.2.1 READ_FKEY

The READ_FKEY module is the initial module used in the Read Function Key subsystem. The purpose of this module is to poll the user input device (the terminal keyboard) for a function key. If a function key is selected within update rate seconds, it will be returned to the invoking module. If a function key is not selected within the update rate period, control is returned to the invoking module with appropriate status. Since structured design techniques have been used, this module does not, and should not, be responsible for any function key validations. Structured design methodology dictates that each black box have one, and only one, function. Therefore, this module simply reads the next keystroke and reports the value to its caller. The actual reading of the keystroke is accomplished by using the screen management routine SMG$READ_KEYSTROKE with an appropriate virtual keyboard identifier.

4.3.2.2 PROCESS_FKEY

The PROCESS_FKEY module receives as input the user-selected function key that was acquired by the READ_FKEY module. It is the responsibility of this module to validate the function key and determine its meaning within the DISKMGR utility. This module returns a status code to its caller designating the classification and validity of the function key. For example, a user may select either the carriage return, the zero key on the keypad, the next screen key, or the previous screen key, in order to signal the DISKMGR utility to change form. It is the intent of this module to process any of these

function keys, but always return a designated code to the caller that means change form. With this design, the process of allowing an additional key to also mean change form is dedicated to this module; where all function keys are defined. In any event, this module returns a status code to its caller detailing the classification and validity of the selected function key.

4.3.2.3 CHANGE_FORM

This module is invoked when the change form function key is selected by the user. The main purpose of this module is to configure internal DISKMGR variables such that the next invocation of the Update Screen subsystem will result in a new form being displayed. To achieve this goal, the CHANGE_FORM module performs two distinct steps.

First, the current form on the display is erased. This step is necessary to insure that data occlusion does not occur. The second step is used primarily to change the internal variable FORM to its new value. However, when the display changes from form number one to form number two, which device should be displayed? The correct action is to display the device currently selected by the token on form number one. Therefore, an additional step is needed to set the correct device when changing to form number two.

4.3.2.4 EXECUTE_COMMAND

This module is invoked when the execute command function key is selected by the user. The main purpose of this module is to provide a means for executing existing VMS utilities to enhance the usefulness of

the DISKMGR utility. To achieve this goal, each field on the form has been associated with a standard VMS command (see appendix C). By selecting the execute command function key, the EXECUTE_COMMAND module will initiate the following algorithm.

First, the field's associated VMS command is checked to determine if a device name parameter is required (SHOW DEVICE/FILES requires the user to specifiy a specific device). If the device name is required in the command string, it is added at this time. Next, the screen management routines are requested to save the current physical screen description. This is recommended practice when performing output with routines that are not using the screen management routines. After the screen description is saved, the VMS command for the particular field is executed by using the Run-Time Library routine LIB$SPAWN. LIB$SPAWN actually creates a subprocess that executes the given VMS command (Digital, "VAX/VMS Run-Time Library Routines Reference Manual"). When the subprocess terminates, control is returned to the EXECUTE_COMMAND module. Finally, a screen management call to restore the physical screen definition is executed and control is returned to the invoking module.

4.3.2.5 REFRESH_SCREEN

This portion of the subsystem is actually a few lines of code in the main body of the DISKMGR utility. When the refresh screen function key is selected by the user, a call is made to the Update Screen subsystem to redraw the entire display. The purpose of including this discussion within the Read Function Key subsystem is to provide

consistency by keeping most function key processing linked to other function key processing. The exception to this rule is the User Help subsystem which also depends on function key input. However, the User Help subsystem is of such importance to the DISKMGR utility that it clearly has earned its right to be a separate subsystem.

4.3.2.6 UPDATE_TOKEN

This module is invoked whenever the next or previous field function keys are selected by the user. The main purpose of this routine is to update the position of the token that is used to identify the current field. To accomplish this task, the current display position of the token must be erased. Then, depending on the function key selected, the token is moved. If next field is selected, the token is advanced one position with care taken to wrap-around if the token moves past the last field on the form. Similarly, if previous field is selected, the token retreats one field position with wrap-around if the token moved past the first field on the form. In essence, this module could be part of the Update Screen subsystem, but, it has been placed in this subsystem for consistency.

4.4 USER HELP SUBSYSTEM

The User Help subsystem provides on-line help messages to interactive users of the DISKMGR utility. This section looks at this subsystem in detail.

4.4.1 Creating And Maintaining A Help Library

One of the major advantages of the DISKMGR utility is the accessibility of information. The DISKMGR utility attempts to make important disk management data available from within one utility rather than forcing the user to navigate several utilities to gather the required data. Coupled with this issue is the desire that once the data is gathered, the user should be able to interpret it. All too often, a disk system manager may gather volumes of data, but its meaning will be unclear. The DISKMGR utility eliminates some of this frustration by providing interactive help text for every displayable field. The following paragraphs discuss the use of VMS help libraries and some of the benefits they provide (for a more detailed discussion, the interested reader is refered to Digital, "VAX/VMS Librarian Reference Manual").

Help libraries contain modules of help text that provide users with information about a program. A user may retrieve help text at the VMS command level (see chapter III) or by using a set of Librarian routines that are callable from many languages. Any utility that is to be considered complete, should have a help library for use by interactive users. The DISKMGR utility provides a help library that an interactive user may browse from the VMS command level, and, this same help library is also manipulated by the user at run-time to provide field specific help. Figure 4-7 is the display shown when a user invokes the DISKMGR help library from the VMS command level.

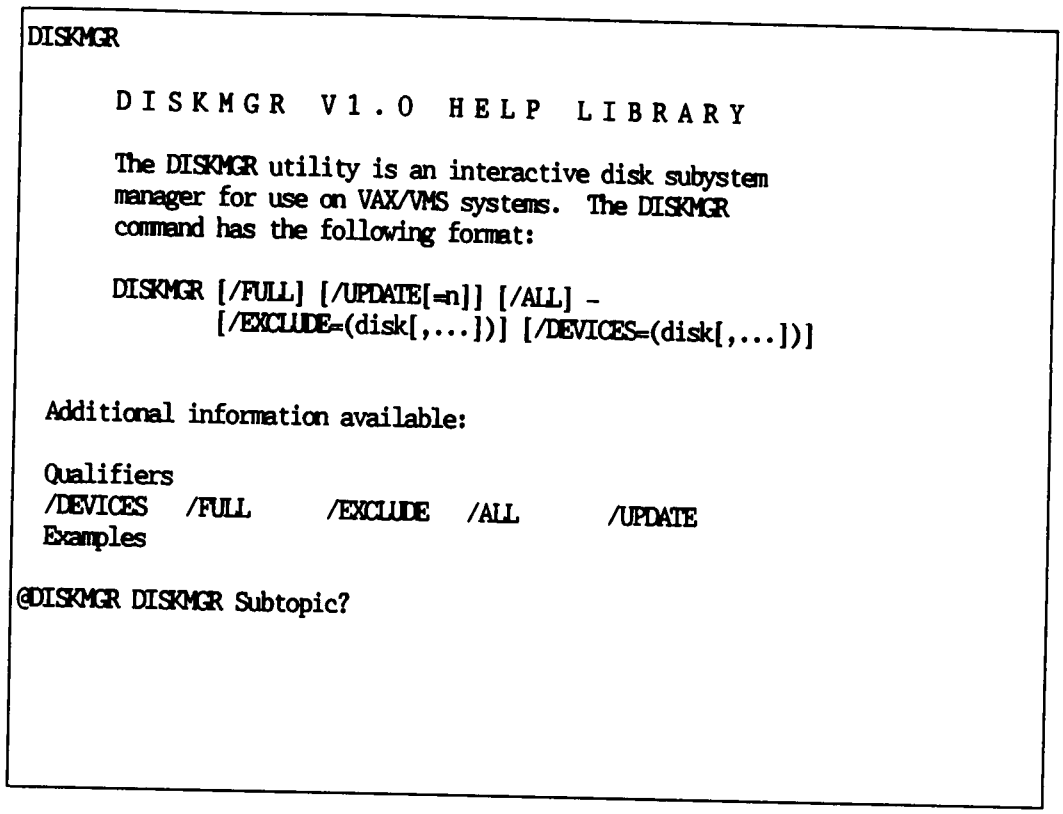


Figure 4-7. VMS Help Screen - DISKMGR

Basic manipulations involved with creating and maintaining the DISKMGR help library may be done from the VMS command level. To create the help library, a file, MS_HELP:DISKMGR.HLP, describing all of the desired help modules must be created. The commands shown in figure 4-8 create the DISKMGR help library and make it accessible from the standard VMS HELP command.

```
$ LIBRARY/CREATE/HELP MS_HELP:DISKMGR.HLB
$ LIBRARY MS_HELP:DISKMGR.HLB MS_HELP:DISKMGR.HLP
$ DEFINE HLP$LIBRARY MS_HELP:DISKMGR
```

Figure 4-8. Help Library Creation and Definition

The first command above creates a file, DISKMGR.HLB, which is the actual help library. The second command defines a logical name which allows the VMS HELP command to locate a new user-supplied help library. (The format of the help text file mentioned above is detailed in Digital, "VAX/VMS Librarian Reference Manual.")

4.4.2 Manipulating A Help Library From FORTRAN

VMS also provides a means to manipulate help libraries from FORTRAN. The Librarian utility that comes with VMS provides a set of routines that manipulate a help library in many ways. These routines provide the capability to create, open, and delete libraries, as well as providing capabilities to read, write, and modify data in libraries. Their basic use in the User Help subsystem is to retrieve a help module from the help library. The actual calls to the help library routines are performed in the module GET_FIELD_HELP described later.

4.4.3 Modules Of The Subsystem

The main purpose of the User Help subsystem is to provide help to the interactive user. Help is provided by the DISKMGR utility for two distinct purposes. The first purpose has been discussed above, that

is, when data is being displayed, the user needs a quick and reliable way to get help on a particular field. The second purpose of providing help to the user is to define the layout of the function keys on the keypad. The procedure to invoke these two types of help displays is discussed in the following module abstracts. The actual use of function keys to signal the User Help subsystem does overlap with the Read Function Key subsystem, but since user help is such an important piece of the DISKMGR utility, this functionality has been extracted from the Read Function Key subsystem. Figure 4-9 is a structure chart for the User Help subsystem.

4.4.3.1 HELP

This module is invoked when either of the help function keys (PF2 or PF3) are selected. This module then determines which of the two function keys was selected and invokes the appropriate module. Within the DISKMGR utility, the PF2 function key implies that the user is requesting general help on the use of the keypad. The PF3 function key is used for obtaining field help.

4.4.3.2 GENERAL_HELP

When the user selects general help, a diagram depicting the setup of the predefined function keys on the keypad is displayed. Since this display is constant, only a single call to the screen management utility is needed. With this call, the current display is temporarily overwritten with the predefined diagram. When the user has finished

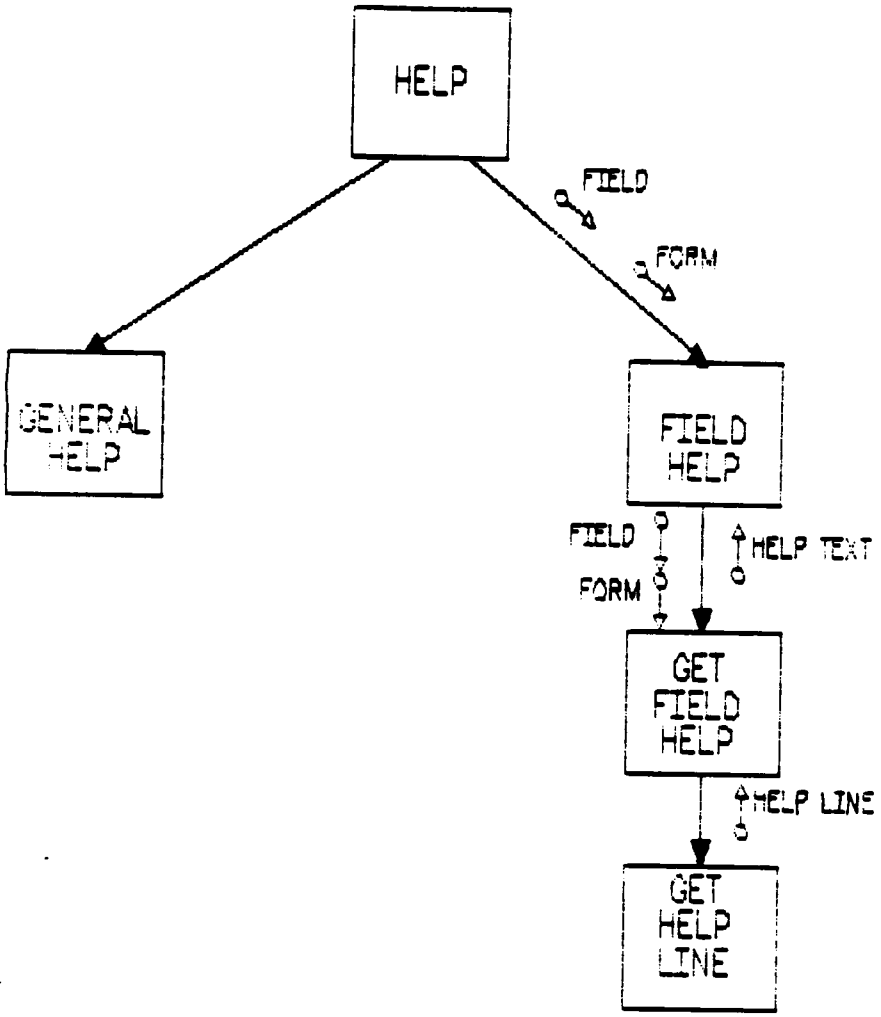


Figure 4-9. User Help Subsystem Structure Chart

viewing the general help display, the diagram is removed and the original display reappears (via the magic of VMS screen management). An example of this help screen in shown in figure 4-10.

4.4.3.3 FIELD_HELP

When the user selects field help, a brief help text message is displayed on the bottom half of the current form. The help text

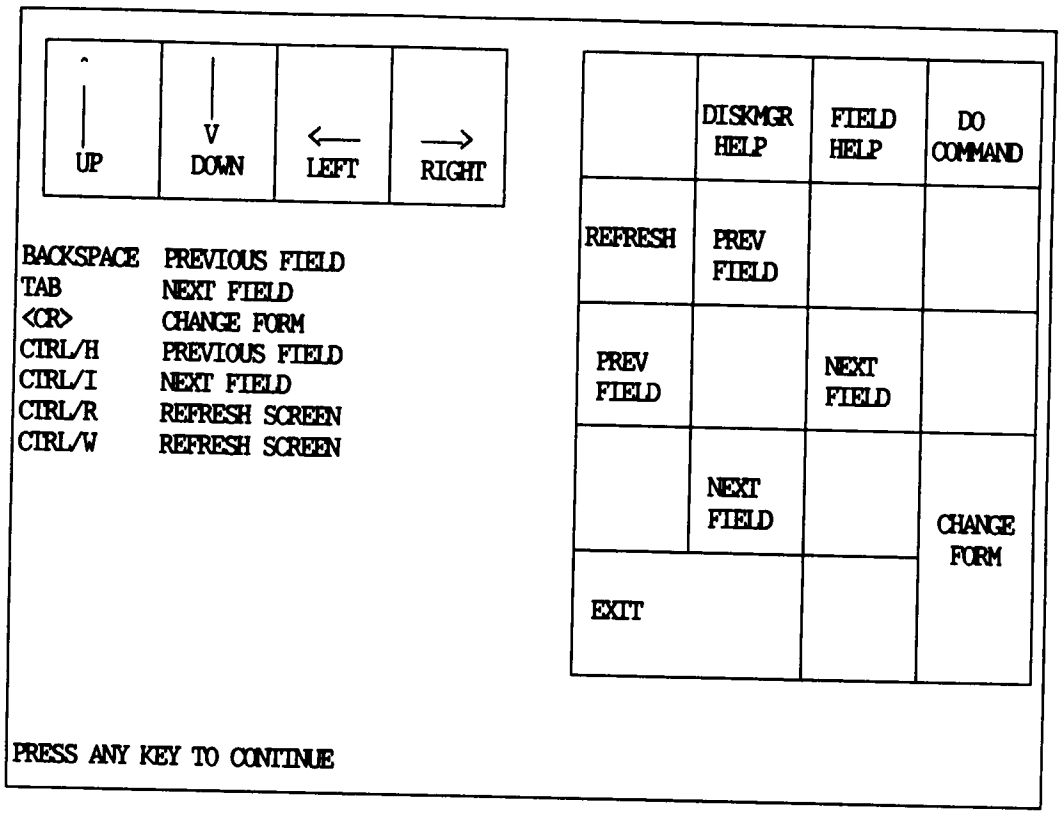


Figure 4-10. General Help Diagram

message corresponding to the current field is retrieved from the help library. The actual retrieval of the help text is handled by modules that are discussed later. This module is responsible for updating the display with the retrieved help text, and then removing the help text when the user is ready to continue. An example of a display after a field help selection is shown in figure 4-11.

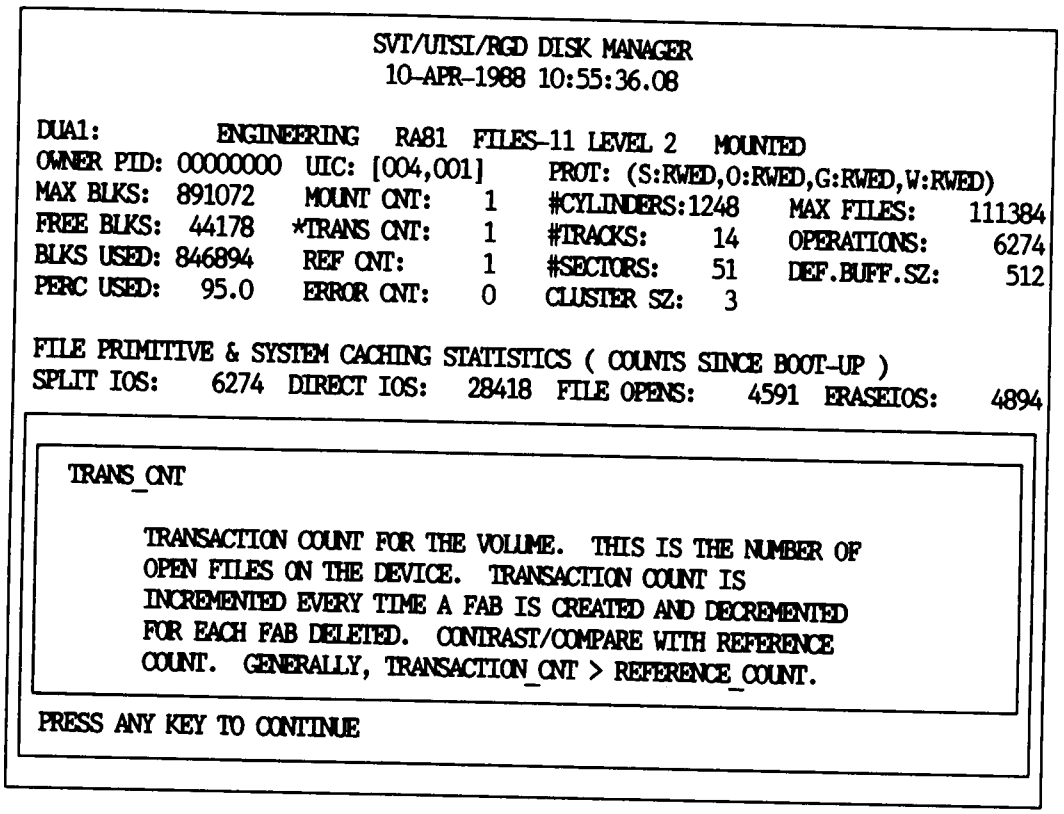


Figure 4-11. Field Help Display

4.4.3.4 GET_FIELD_HELP

This module and the GET_HELP_LINE module work as a team to retrieve text from the help library. In trying to understand the method of retrieving text from a help library, the Librarian routine LBR$OUTPUT_HELP must be examined a little more closely.

GET_FIELD_HELP invokes LBR$OUTPUT_HELP passing GET_HELP_LINE as the address of a subroutine to receive the help text. Also, the field name of the current token position, which corresponds to the key value in the help library, is passed to the Librarian routine along with the appropriate library name. When the call completes, GET_HELP_LINE has inserted into an array of character strings, the retrieved help text. This help text is then returned to FIELD_HELP in order to be displayed.

4.4.3.5 GET_HELP_LINE

GET_HELP_LINE receives as input from the LBR$OUTPUT_HELP routine, a character string representing a line from the help library. This module then inserts the line into an array of character strings that will eventually contain all of the help text for the current field. This module is unique in that it is called indirectly by GET_HELP_LINE.

4.5 GET DEVICE INFORMATION SUBSYSTEM

The Get Device Information subsystem provides all the necessary device data points to the DISKMGR utility. This section looks at this subsystem in detail.

4.5.1 Using VMS System Services

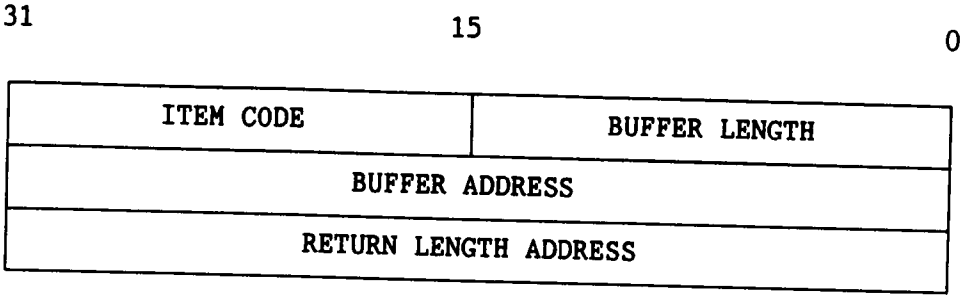
As mentioned earlier, one of VMS' most admired attributes is its great assortment of utilities available to all users. A few of these utilities have been discussed under previous subsystems. This subsystem uses some routines that have a special name; system services. In general, a VMS system service is a routine that returns information

related to some process, device, or other aspect of the system (for a more detailed discussion of all the possible system service calls, refer to Digital "VAX/VMS System Services Reference Manual"). All system services begin with an SYS facility abbreviation when used in FORTRAN. Furthermore, all system services are written as functions where the function value returned is a status code that can be used to detect the success or failure of a system service call. System services play an important role in the Get Device Information subsystem.

4.5.2 Modules Of The Subsystem

The DISKMGR utility employs the SYS$GETDVIW system service to return information on the status of devices connected to the system. The scope of the SYS$GETDVIW system service has been narrowed even further by the DISKMGR utility, in that only disk device information is requested.

The SYS$GETDVIW system service returns device dependent information to its caller. The information returned to the caller is defined through a structure known as an itemlist. An itemlist is a set of pointers and addresses for defining the information to be returned and where the returned information is to be stored. It is through the use of the itemlist data structure that many VMS system services receive and return data. For more information on the layout of the itemlist structure, figure 4-12 will prove useful.




```
STRUCTURE   /ITMLST/
   UNION
      MAP
         INTEGER*2       BUFFER LENGTH
         INTEGER*2       ITEM CODE
         INTEGER*4       BUFFER ADDRESS
         INTEGER*4       RETURN LENGTH
      END MAP
      MAP
         INTEGER*4       END LIST
      END MAP
   END UNION
END STRUCTURE
```

Figure 4-12. Itemlist Data Structure in FORTRAN

As comprehensive as the list of VMS system services seems, the development of the DISKMGR utility determined a very noticable hole. There does exist a need for a system service that would return information regarding system performance. Such a service, perhaps SYS$GETSPI, could reach into the executive data areas of VMS and return performance data that is already being maintained by VMS. However, such a routine does not exist (yet), and these data values must be retrieved by a more brute-force method (see module GET_PMS_DATA). The modules of the Get Device Information subsystem are shown in figure 4-13.

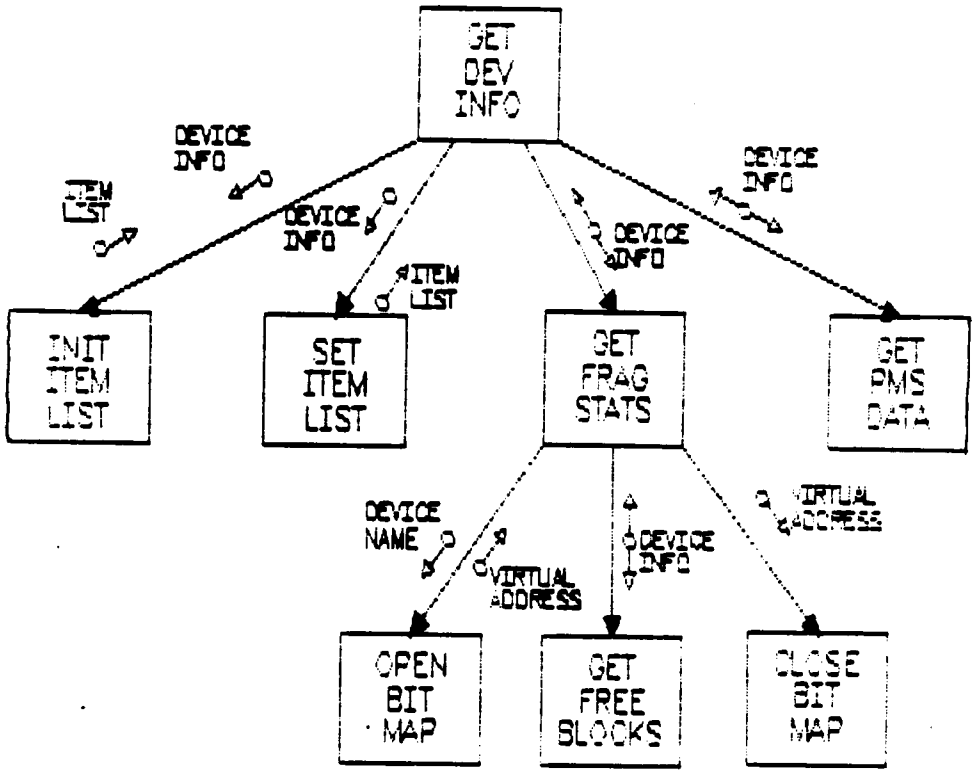


Figure 4-13. Get Device Information Subsystem Structure Chart

4.5.2.1 GET_DEV_INFO

The GET_DEV_INFO module is the controlling module of this subsystem and is responsible for gathering all device specific information. That is, this module will make all of the calls necessary to retrieve the appropriate information on each device being monitored. To perform this data gathering, all modules within this subsystem perform a very specific task.

Before the algorithm used by this module can be understood, a few of the major data structres should be considered. The information gathered for each device can be classified as belonging to one of two groups; brief or full. The brief set of data points are composed of all data needed to display form number one. The full set of data points are composed of all data needed to display form number two. When manipulating form number one, the DISKMGR utility must navigate through many copies of the brief set of data points (one for each disk). When manipulating form number two, only one set of brief data points and one set of full data points are needed (only monitoring one disk). The actual structures used to define the brief and full data points are in an include file, MS_INC:DEV_ATT.INC, listed in appendix B.

Now that the major data structures for GET_DEV_INFO have been defined, our attention can now be addressed to the algorithm used to implement this module. To begin with, GET_DEV_INFO initializes the itemlist data structure. This allows the retrieved data to be placed in the correct locations of the device attribute structure (described in the previous paragraph). The next step of the algorithm involves a call to determine which set of data points needs to be retrieve. For example, if form number one is being displayed, there is not a need to retrieve all of the full data points since the full set of data points will not be displayed.

The third and last step of the algorithm retrieves all of the device information to be used on the current form. A call is issued to the VMS system service SYS$GETDVIW to retrieve the data points

available to it. This call is made regardless of the form being displayed. If form number one is being displayed, the algorithm terminates at this point. If form number two is being displayed, the algorithm must retrieve the file system performance statistics and the fragmentation statistics of the specified device. After all needed data points have been gathered and placed in the appropriate data structures, the algorithm terminates.

4.5.2.2 INIT_ITEMLIST

INIT_ITEMLIST is the module responsible for initializing the itemlist data structure. This initialization occurs at the beginning of each cycle through the Get Device Information subsystem. This module defines the destination addresses of where the retrieved data points are to be stored. This module also defines the appropriate return lengths for each selected data point. Upon termination, a status code defining the success or failure of the module is returned to the caller.

4.5.2.3 SET_ITEMLIST

SET_ITEMLIST is the module responsible for optimizing the itemlist. As mentioned earlier, the itemlist consists of all data points to be retrieved by a single call to SYS$GETDVIW. However, if form number one is the current form, and a full set of data points are retrieved, then precious CPU time has been wasted gathering data that will never be displayed. To alleviate this problem, SET_ITEMLIST checks the current form being displayed and then sets the appropriate

end of list marker in the itemlist data structure. By optimizing the number of data points to be retrieved, CPU time is conserved. This module then terminates and returns a function status value to its caller.

4.5.2.4 GET_PMS_DATA

The purpose of this module is to retrieve all of the performance management statistics needed by the DISKMGR utility. The writable executive areas of the VMS kernal consist of many data values used to describe the cumulative system performance. These data values are the same values that are sampled by the VMS Monitor utility for most of its data. For instance, file system performance monitor data may be found that describes the performance of the On-Disk Structure Level 2 Extended QIO Processor (XQP). Most of the data needed by the DISKMGR utility centers around the XQP. (A reader interested in knowing more about the VMS executive data areas should consult Kenah, "VAX/VMS Internals and Data Structures: Version 4.4".)

To retrieve this performance data from the system executive requires a bit of trickery within FORTRAN. First of all, the symbols used to denote the desired values are declared as external. Next, the address associated with each of these values is stored in a local variable. Finally, indirect addressing is employed to retrieve the value stored in the executive. As mentioned earlier, a system service to provide this data would have been useful rather than having to process the data in this manner. When all performance data points have

been gathered, control is returned to the caller along with a status code detailing the success or failure of the routine.

4.5.2.5 GET_FRAG_STATS

GET_FRAG_STATS is the module responsible for retrieving all of the free space fragmentation statistics used by DISKMGR. The following algorithm is employed to determine the disk free space fragmentation status. First, a set of ranges has been predetermined by the DISKMGR utility. These sets of ranges are counters to the number of free blocks on a device. For example, given the range ten to twenty-five, and a count of three, one can be assured that there are three sets of contiguous blocks on the disk with size ten to twenty-five.

GET_FRAG_STATS employs a simple three-step algorithm. In the first step, a call is made to read the specified device's bit map file. Recall from chapter II that the bit map file contains a set of bits representing the availability of each block on the disk. For each set of free blocks on the disk, a call is then made to determine the number of contiguous free blocks. Once the number of contiguous free blocks is known, the appropriate counter can then be incremented. Processing continues in this fashion until there are no more free blocks to be processed. At this stage, the routine terminates and returns both control and a status value to its caller.

4.5.2.6 OPEN_BITMAP

OPEN_BITMAP reads the storage bit map file into an extension of process virtual address space. It then closes the bit map file to

insure that the file is only be locked for a small amount of time. (This module is a macro assembly language program written originally by James B. Fischer and submitted to the Fall 1986 Decus user's public domain tape library. The module has been modified for use by the DISKMGR utility.)

4.5.2.7 GET_FREE_BLOCKS

GET_FREE_BLOCKS processes the contents of the storage bit map file as one long continuous string of bits. To determine the number of free blocks on the disk, one must start at logical block number zero, and find the first free block. Once the first free block has been located, searching continues from this block to find the first allocated block. The number of blocks between the first free block and the first allocated block is the number of contiguous free blocks. It is this value that is returned to the caller. Upon the next invocation of this routine, processing begins with the block immediately following the allocated block found above. (This module is a macro assembly language program written originally by James B. Fischer and submitted on the Fall 1986 Decus user's public domain tape library. The module has been modified for use by the DISKMGR utility.)

4.5.2.8 CLOSE_BITMAP

As mentioned briefly in the OPEN_BITMAP module, the storage bit map file is read from disk into an extension of process virtual address space. This is made possible by using the Run-Time Library routine LIB$GET_VM which allocates virtual memory for use by a program. The

CLOSE_BITMAP module has the responsiblity of releasing the virtual memory when processing of the bit map file has been completed. The memory is released by using the routine LIB$FREE_VM.

4.6 UPDATE SCREEN SUBSYSTEM

The Update Screen subsystem manages the screen appearance for the DISKMGR utility. This section looks at this subsystem in detail.

4.6.1 Managing Screen Appearance

As described in the Read Function Key subsystem, VMS provides a set of Run-time Library routines that are useful for device-independent input and output. In fact, the suggested tools for managing the appearance of the terminal screen on VMS are these SMG Run-Time Library routines (Digital, "Guide to Programming on VAX/VMS"). Before looking at the modules which constitute this subsystem, a brief overview of the screen management routines is necessary.

To use the screen management facility, a user must have a knowledge of the terms pasteboard and virtual display. A pasteboard is a portion of the physical display device used for the output. Virtual displays are logical displays that hold the actual output data to be viewed by the user. To understand this a little more easily, assume the pasteboard is the entire output screen. The screen may be divided into multiple "windows", where each window corresponds to a virtual display. Virtual displays may overlap one another or may be covered entirely by other virtual displays. VMS screen management routines guarantee that when one virtual display covers another, then if the top

virtual display is removed, the previously hidden virtual display will become visible. That is, the screen management routines are much more than just a set of I/O routines. They are truly integrated and do provide total screen management in a mode transparent to the user.

Now that a better understanding of pasteboards and virtual displays has been obtained, this discussion can concern itself with the manipulation of these objects. The screen management routines allow many operations to be performed on pasteboards and virtual displays in order to manage screen appearance. Some very useful routines are ones to create a pasteboard, create a virtual display, view, erase, and read/write data to a virtual display. Most of these functions are utilized by the Update Screen subsystem.

4.6.2 Modules Of The Subsystem

The main purpose of the Update Screen subsystem is to manage the appearance of the user's output screen. The screen management routines mentioned in the previous section provide a consistent and reliable design to the DISKMGR utility. Since these routines are a part of VMS, the Update Screen subsystem will most likely be preserved even with the introduction of new display devices. This reasoning led to the decision to use the screen management routines for all output of the DISKMGR utility. The modules that constitute the Update Screen subsystem are shown in the figure 4-14.

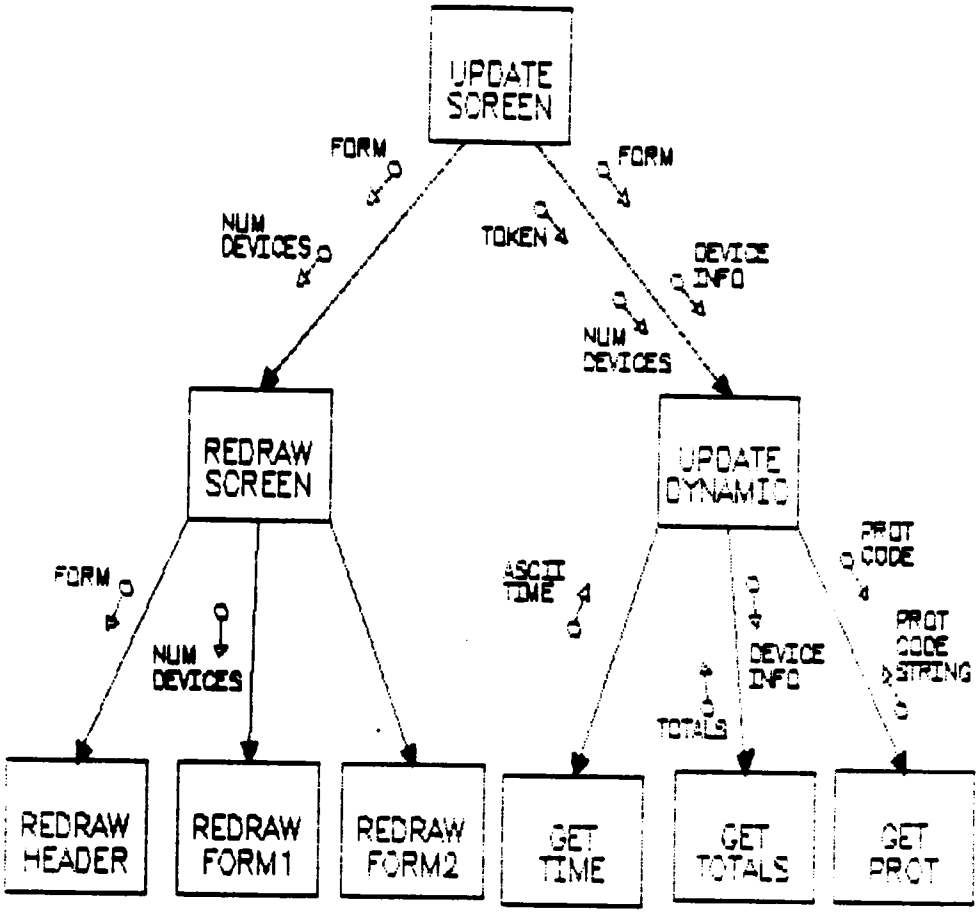


Figure 4-14. Abbreviated Update Screen Subsystem Structure Chart

4.6.2.1 UPDATE_SCREEN

The UPDATE_SCREEN module serves as the controlling module for most of the Update Screen subsystem. The purpose of this module is to update the display being viewed by the user. To understand what must be accomplished when updating the display, a look inside this module is worthwhile. First, consider the display has been partitioned into a background and a foreground. The background of the display is all of

the text necessary to label the fields. The background is static in that it does not need to be updated except when the change form function key or the refresh screen function key has been activated. Meanwhile, the foreground of the display consists of all the actual data being displayed. This information is dynamic and must be updated during every cycle through the system. Understanding this concept is a pre-requisite to understanding this subsystem.

The algorithm for UPDATE_SCREEN begins by determining if the change form or refresh screen function keys have been activated. If either of these function keys are activated, the entire background and foreground will be redrawn. Regardless of any function key activations, the dynamic portion of the display is always updated. When the update routines have been completed, this module terminates. Control and a status value are returned to the caller.

4.6.2.2 REDRAW_SCREEN

REDRAW_SCREEN is the module responsible for redrawing all of the background text. Whenever the display needs to be redrawn, this module is invoked to redraw the static portion of the display. This is accomplished in the following steps. First, the header of the display must be redrawn. The header consists of the first few lines on either form. The second step must redraw the remaining portion of the display based on the current form being displayed. The module then terminates, returning control and a status value to its caller.

4.6.2.3 REDRAW_HEADER

This module is responsible for redrawing the header portion of the current form. Since this module is the first step involved in redrawing the entire display, initial display erasure is also performed by REDRAW_HEADER. The header consists of the first three lines of the display where the first line is the title or banner for the DISKMGR utility. The second line of the header contains the current date and time display and the third line is a blank line that serves as a separater for the header and actual form. Upon termination of REDRAW_HEADER, control is returned to the invoking module. A status code representing the success or failure of the module is returned as the function's value.

4.6.2.4 REDRAW_FORM1

This module is responsible for redrawing the background text associated with form number one. This text includes the header for each of the columns, an appropriate number of blank lines (one for each device), and a total subheader at the bottom of selected columns. Upon termination, a status code is returned to the caller.

4.6.2.5 REDRAW_FORM2

This module is responsible for redrawing the background text associated with form number two. This text includes appropriate field labels for each value to be displayed on the form. Upon termination, a status code is returned to the caller.

4.6.2.6 UPDATE_DYNAMIC

The UPDATE_DYNAMIC module is responsible for updating the dynamic portion of the display. To accomplish its task, all data points must be updated, as well as, the system time in the header. This module knows the format of each form and the desired output location for all specified values. One of the problems in providing display updates is that the screen management routines require all data to be in character format. Therefore, all integer and real data points must be converted to character data points before they can be displayed. However, dynamic display updating does not occur as a series of physical writes to the terminal, but, rather a series of writes to the virtual display. By forcing the screen management routines to buffer the display output, only one physical write to the terminal is generated. This technique is much more efficient for full display updates and should be considered by any routine performing display updates. This module returns a status code detailing the success or failure of the display updates.

4.6.2.7 GET_TIME

This module is responsible for obtaining the current system date and time in character format. This date and time character string is then returned to UPDATE_DYNAMIC and integrated into the header portion of the physical display. The current date and time is acquired by using the VMS system service SYS$GETTIM which returns an eight-byte structure. This structure is then passed to another system service, SYS$ASCTIM, which provides the module with a twenty-three byte

character string of the form "DD-MMM-YYYY:HH:MM:SS.CC". It is this character string that is actually returned to UPDATE_DYNAMIC.

4.6.2.8 CREATE_BRIEF_LINE

CREATE_BRIEF_LINE is a module invoked by UPDATE_DYNAMIC whenever form number one is being updated. The purpose of this module is to format a character string in such a manner that it represents one line of data for a given device on form number one. To do this, UPDATE_DYNAMIC provides this module with the device's set of brief data points. CREATE_BRIEF_LINE then formats these data points into a character string that contains the data points in the appropriate columns. When formatting completes, a status code, as well as the formatted output string, are returned to the caller.

4.6.2.9 GET_TOTALS

GET_TOTALS is a module invoked by UPDATE_DYNAMIC whenever form number one is being updated. The purpose of this module is to calculate the values to be displayed on the total line of form number one. To accomplish this task, GET_TOTALS is provided with a list of all the brief data points for every disk being monitored. Appropriate data points, such as device errors and number of free blocks, are summed. These summations are then returned to UPDATE_DYNAMIC.

4.6.2.10 CREATE_TOTAL_LINE

CREATE_TOTAL_LINE is a module invoked by UPDATE_DYNAMIC whenever form number one is being updated. The purpose of this module is to

format a character string in such a manner that the character string will represent the total line displayed at the bottom of form number one. To do this, UPDATE_DYNAMIC provides this module with a set of totals created by GET_TOTALS. CREATE_TOTAL_LINE then formats these totals into a character string that contains the desire output in appropriate columns. When formatting completes, a status code, as well as the formatted output string, are returned to the caller.

4.6.2.11 OUTPUT_TOTAL_LINE

This module is responsible for displaying the total line for form number one. The total line to be output has been created by the module CREATE_TOTAL_LINE and is assumed to be properly formatted for output to the display. This module returns a status code to its caller, UPDATE_DYNAMIC, detailing the success or failure of the display update.

4.6.2.12 OUTPUT_CHARS

OUTPUT_CHARS is a general purpose module used by several other modules in the Update Display subsystem. Its function is to output a character string value of a specified length onto a virtual display. The output will occur at a specified row and column offset in the virtual display. An appropriate status value is returned to the caller.

4.6.2.13 OUTPUT_INTEGER

OUTPUT_INTEGER is also a general purpose module used by several other modules in this subsystem. Its function is to output a specified

integer value onto a virtual display. The integer value may optionally be zero filled within a field and is output at a specified row and column offset in the virtual display. An appropriate status value is returned to the caller.

4.6.2.14 INSERT_INTEGER

INSERT_INTEGER is a general purpose module used quite frequently by many other modules in the Update Screen subsystem. Its purpose is to convert an integer value to a character string suitable for output to a virtual display. The length of the output value is passed as a parameter into the module. If the length parameter is less than zero, then the character string returned will be padded with leading zeroes. An appropriate status value is returned to the caller.

4.6.2.15 OUTPUT_REAL

OUTPUT_REAL is a general purpose module used by several other modules in this subsystem. Its function is to output a specified real value onto a virtual display. The real value may be displayed with any precision to the right of the decimal point by supplying the desired value as the resolution parameter. Output occurs to a specified row and column offset in the virtual display. The width of the output field is controlled by the length parameter passed to the module. An appropriate status value is returned to the caller.

4.6.2.16 INSERT_REAL

INSERT_REAL is also a general purpose module used quite frequently by many other modules in the Update Screen subsystem. Its purpose is to convert a real value to a character string suitable for output to a virtual display. The length of the output value is passed as a parameter into the module as well as the desired resolution. An appropriate status value is returned to the caller.

4.6.2.17 GET_PROT

One of the attributes associated with every device is its protection code. The protection code is used by VMS to define the class of users which may access the device. When the device protection data point is retrieved in the Get Device Information subsystem, it is retrieved as a four-byte structure. It is the responsibility of this module to convert the acquired four-byte structure into a format understood by most VMS users.

The first four bits of every byte define the access protection for each of the four VMS access modes (system, owner, group, and world). Within each byte, bit zero is the read code (R), bit one is the write code (W), bit two is the execute code (E), and bit three is the delete code (D). Access is granted if the desired bit is cleared. Knowing the format of this standard VMS protection code mask allows this module to reformat the protection code into a standard format for displaying VMS protection codes. This format is shown below.

(S:RWED, O:RWED, G:RWED, W:RWED)

Once the string has been formatted completely, it is returned to the caller along with a status code detailing the success or failure of the module. (A user interested in knowing more about the use of VMS protection codes should consult Digital, "Guide VAX/VMS to System Security" for a complete discussion.)

4.7 INITIALIZATION AND ERROR PROCESSING MODULES

The initialization and error processing modules have been grouped together to form the sixth subsystem of the DISKMGR utility. These modules are not a true subsystem since they do not have a clear, concise function to perform. These modules, however, perform many varying functions for the benefit of the first five subsystems.

4.7.1 DISKMGR Initializations

The initialization modules of the DISKMGR utility have the unique task of performing initialization setup for all other subsystems. Initialization in this fashion allows the other subsystems to operate in a mode oblivious to the real world. Each of the modules defined below have a set of responsibilities that must be performed correctly if the entire DISKMGR utility is to function properly. The following paragraphs define each step in the DISKMGR initialization process.

4.7.1.1 INIT_IO

The DISKMGR utility can hardly operate unless some explicit setup for input and output has been done. The INIT_IO module is responsible for creating the many virtual displays used by the DISKMGR utility as

well as creating the pasteboard and virtual keyboard necessary for input and output. The following virtual displays are created by INIT_IO.

1. Form Number One

2. Form Number Two

3. General Help

4. Field Help

5. Message Board

Form number one virtual display is used to hold the display representing form number one. Likewise, form number two virtual display is used to hold form number two's display. The general help virtual display is used to hold the keypad layout while the field help virtual display is used to hold the field help display. Finally, the message board virtual display is used to hold error messages generated by DISKMGR.

In addition to creating the above virtual displays, INIT_IO also creates the pasteboard and defines the virtual keyboard. The pasteboard and virtual keyboard are both mapped to the SYS$OUTPUT device (usually the user's terminal). In order to operate the DISKMGR utility, SYS$OUTPUT must translate to a device suitable for input and output. Once all of the virtual displays, pasteboards, and keyboards have been created, the INIT_IO routine is terminated.

4.7.1.2 START_IO

The function of the START_IO module is to paste an initially blank virtual display onto the pasteboard. To do this, the current form must have been determined from the command line. This function allows the DISKMGR utility to begin processing without having to issue an unnecessary call to CHANGE_FORM before its first loop.

4.7.1.3 INIT_HELP

The function of the INIT_HELP module is to initialize the general help display. This display is constant and it is never modified by the DISKMGR utility. Therefore, to avoid delay when the user selects the general help function key (PF2), the virtual display is defined during initialization and simply must be pasted to the pasteboard during run-time. This function allows the User Help subsystem to do its chore in a more efficient manner.

4.7.1.4 INIT_COMMANDS

INIT_COMMANDS is a module that maps a specified VMS command to each field. The mapping of the commands to the field occurs during initialization so that needless delay can be avoided when the execute command function key is activated. This capability allows the Read Function Key subsystem to execute without performing some type of table lookup. For a complete list of field names and their corresponding commands, see Appendix C.

4.7.1.5 INIT_TOKEN

INIT_TOKEN is the module that is responsible for predetermining the initial location of the token on the display. The token always begins at the first field of the form, but, depending upon the command line entered by the user, the initial form may vary. Therefore, this module determines the initial location of the token by examining the initial form selected by the user.

4.7.2 Error Processing And The Message Utility

Another of VMS' advantages comes from the ability of users to supplement system error condition codes and messages with their own set of codes and messages. This is made possible by the Message Utility which is available to all VMS users (Digital, "VAX/VMS Message Utility Reference Manual). To utilize this capability, a user must create a message source file. This message source file contains a list of condition codes and associated error messages. These message codes are then compiled by the Message Utility and linked to the user's program. Figure 4-15 shows the message source file used by the DISKMGR utility.

4.7.2.1 REPORT_EXCEPTION

REPORT_EXCEPTION is responsible for displaying all error messages to the user. This module receives as input a condition code which is defined by the Message Utility discussed above. This code is then given to the VMS system service SYS$GETMSG. SYS$GETMSG returns to REPORT_EXCEPTION the text associated with the given error code.

```
.FACILITY DISKMGR, 1/PREFIX=DISKMGR__
.SEVERITY WARNING
   ERRUNPDIS "Could not unpaste virtual display.  "
   DEVNOTEXI "Specified device does not exist.  "
   UNAVALDEV "Unable to validate specified device.  "
   DEVNOTDIS "Specified device is not a disk.  "
   ERRSAVSCR "Error saving current screen display.  "
   ERRSPAWN  "Error spawning DCL command.  "
   ERRRESSCR "Error restoring screen display.  "
   MAXDEVEXC "Maximum number of devices exceeded.  "
   GETDVIERR "$GETDVI returned error.  "
   ERRGETIME "Error obtaining system time.  "
   ERRCVTIME "Error converting time to ASCII.  "
   ERRCREPBD "Error creating pasteboard.  "
   ERRCREDIS "Error creating virtual display.  "
   ERRCVTVAL "Error converting value.  "
   ERRPUTCHR "Error writing character to display.  "
   ERREADKSK "Error reading keystroke.  "
   ERRREMDIS "Error unpasting virtual display.  "
   FRAGERROR "Error reading BITMAP file.  "
.END
```

Figure 4-15. Message File For DISKMGR Utility

The text returned from SYS$GETMSG is then placed in an output message that details the module in which the error occurred. This complete output message is then written to the message board virtual display.

4.7.2.2 CLEAR_ERROR

CLEAR_ERROR is a module that has the responsiblity to clear the message board virtual display. The message board virtual display is cleared after every ten screen updates and a blank message board virtual display is repasted to the pasteboard.

4.8 SUMMARY

In this chapter, a complete discussion of every module involved in the DISKMGR utility has been presented. The basic workings of each module have been discussed, not to show how the utility as a whole works, but rather to show the internal workings and data structures of each module. (Note, this chapter is intended to be a programmer's reference guide to the DISKMGR utility. For a discussion on the DISKMGR utility from a user's viewpoint, please consult chapter III.) Also presented in this chapter is an overview of the many VMS utilities used by DISKMGR. These include the command line interface (CLI) routines, the screen management (SMG) routines, and many VMS system services (SYS). This chapter also presented an introduction on the use of help libraries and the message utility as well. However, all of these utilities were presented for the reader's information and the discussions were centered on how DISKMGR manipulates them. For discussions on the general use of these utilities, many references were given throughout the chapter.

CHAPTER 5

FUTURE RECOMMENDATIONS

The desire of this thesis, to provide a utility for total disk system management, has been accomplished with a reasonably high degree of success. DISKMGR is a utility which is capable of gathering useful disk management statistics, but, at the same time, it is simple and easy to use. This chapter discusses the success of DISKMGR and recommends future enhancements to the utility.

5.1 DISKMGR'S ACCOMPLISHMENTS

DISKMGR began with a simple goal: provide a disk management utility that is easy to operate and easy to understand. To reach this goal, DISKMGR was designed to be a single source utility which would gather important disk management statistics on behalf of the user. Also, DISKMGR should provide the user with a means to interpret and understand the gathered data. This will enable the non-professional user to make professional disk management decisions in a more productive manner. Moreover, DISKMGR should not simply integrate disk management data from existing VMS utilities. It should provide data

points that can not be determined by any other existing VMS utility as well. All in all, these goals have been met and the DISKMGR project has been a successful venture.

5.2 FUTURE DIRECTIONS

As expressed in chapter II, proper VMS disk management is a process that requires knowledge about the entire disk storage subsystem. That is, management of the disk storage subsystem has many facets. DISKMGR has addressed several of these topics, but there are some other techniques that need to be developed.

One future enhancement for the DISKMGR utility is an ability to record and playback disk management statistics. This capability would provide the user a means to record disk management statistics on a regularly scheduled interval. Once the data has been recorded, it could be played back as a means to compare disk activity in various intervals. By using the recording feature, the user may be more capable of better load balancing of applications among multiple disk drives.

Another future enhancement to the DISKMGR utility that would prove to be very beneficial is the capability of performing data analysis. This feature would allow DISKMGR to monitor a disk and provide its user with recommendations that could enhance disk performance. For example, this feature would enable DISKMGR to set an alarm condition whenever the fragmentation factor of a disk exceeded a certain level. This alarm condition would then notify the user to take appropriate action to correct the problem. Or perhaps, DISKMGR could monitor the file

system performance statistics, calcuating hit/miss ratios, and alert the user when memory cache sizes need to be adjusted.

Also within the spectrum of possible enhancements, a feature could be added to signal all VMS disk users when free disk space falls below a specified value. This would free the user from periodically monitoring disk space allocation since DISKMGR would forewarn all disk users of possible space allocation problems. An additional feature, in this same category, would include the ability to detect when the number of disk errors exceeds a certain value. In this case, DISKMGR may be able to warn the user prior to any major files being damaged or lost. Yet another enhancement within this class could be a feature developed to determine when the number of operations to a disk is exceeding the capability of the disk. Again, this feature would be very useful in load balancing among various disk drives. There are many numerous features of this type that could be very useful to a disk manager.

Finally, a future recommendation for DISKMGR concerns the pending release of the new design of the VMS system management architecture. As mentioned in chapter I, the VMS designers are currently developing a set of modularized routines for performing system management. When these routines are introduced, DISKMGR should seize the opportunity to manipulate them for its benefit. Hopefully, with the aid of these routines, DISKMGR can extend its current limitations and progress into a complete utility capable of unlimited growth.

5.3 CONCLUSION

In conclusion, the goals and expectations of this thesis have been met with the best of intentions. A complete disk management utility would take many months of dedicated development to meet all of the requirements associated with disk management. Disk management has been shown to be a very complicated task, but, a task that can be accomplished very efficiently when it is properly understood and its limitations are respected.

This thesis has attempted to provided some interesting insights on issues relating to disk management within VMS. DISKMGR has exposed some interesting problems within the current VMS system management architecture and has demonstrated better means to reached the desired results. This thesis has provided a great deal of satisfaction to its author and, given the above observations, has certainly been worth the time and effort involved during its preparation.

BIBLIOGRAPHY

BIBLIOGRAPHY

Digital Equipment Corp. Guide to Programming on VAX/VMS. VMS Version 4.4. Bedford: Digital Press, 1986.

Digital Equipment Corp. Guide to Using DCL and Command Procedures on VAX/VMS. VMS Version 4.0. Bedford: Digital Press, 1984.

Digital Equipment Corp. Guide to VAX/VMS Disk and Magnetic Tape Operations. VMS Version 4.4. Bedford: Digital Press, 1986.

Digital Equipment Corp. Guide to VAX/VMS Performance Management. VMS Version 4.2. Bedford: Digital Press, 1986.

Digital Equipment Corp. Guide to VAX/VMS System Security. VMS Version 4.2. Bedford: Digital Press, 1985.

Digital Equipment Corp. VAX/VMS Command Definition Utility Reference Manual. VMS Version 4.0. Bedford: Digital Press, 1984.

Digital Equipment Corp. VAX/VMS DCL Dictionary. VMS Version 4.4. Bedford: Digital Press, 1986.

Digital Equipment Corp. VAX/VMS Librarian Reference Manual. VMS Version 4.0. Bedford: Digital Press, 1984.

Digital Equipment Corp. VAX/VMS Message Utility Reference Manual. VMS Version 4.0. Bedford: Digital Press, 1984.

Digital Equipment Corp. VAX/VMS Run-Time Library Routines Reference Manual. VMS Version 4.4. Bedford: Digital Press, 1986.

Digital Equipment Corp. VAX/VMS System Services Reference Manual. VMS Version 4.4. Bedford: Digital Press, 1986.

Ellis, Bruce. "Writing VMS Performance Tools." DECUS VAX Systems Session Notes. Marlboro: DECUS, 1987.

Fisher, Jim. "Advanced System Management Techniques." VAX Professional. Volume 10, Number 1. Spring House: Professional Press, Inc., 1988.

Fleischman, Wef. "ODS-2 Disk Optimization." USA DECUS 1987 Spring Proceedings. Marlboro: DECUS, 1987.

Kenah, Lawrence J., Ruth E. Goldenberg, and Simon F. Bate. VAX/VMS Internals and Data Structures: Version 4.4. Bedford: Digital Press, 1988.

Page-Jones, Meilir. The Practical Guide to Structured Systems Design. Englewood Cliffs: Yourdon Press, 1980.

Sun, Moses. "Disk I/O: Part 1." DEC Professional. Volume 6, Number 11. Spring House: Professional Press, Inc., 1987.

Sun, Moses. "Disk I/O: Part 2." DEC Professional. Volume 6, Number 12. Spring House: Professional Press, Inc., 1987.

APPENDIXES

APPENDIX A

DISKMGR SOURCE LISTINGS

This appendix contains a complete source listing of the DISKMGR utility. The main module, DISKMGR.FOR, is listed first with all other modules listed alphabetically.

A.1 DISKMGR

```
        PROGRAM         DSKMGR
C
C       name:           Disk Manager
C
C       abstract:       Interactive VMS disk subsystem manager.
C
C       history:        11/01/87---pdl by Randy G. Dotson
C                                  Done for Master of Science Degree
C                                  at the University of Tennessee Space Institute.
C
C       includes:
C
        INCLUDE         '($SSDEF)'                      ! return status definitions
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
        INCLUDE         'MS_INC:DEV_ATT.INC'
                        ! device attribute structure definition
        INCLUDE         'MS_INC:DISK_ANAL_INIT.INC'     ! initialization definitions
        INCLUDE         'MS_INC:DISK_ANAL_CONST.INC'    ! constant definitions
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! IO definitions
        INCLUDE         'MS_INC:ITMLST_STRUCT_DEFN.INC' ! itemlist structure definition
        INCLUDE         'MS_INC:TOKEN_DEFN.INC'         ! token definitions
C
C       globals:
C

C
C       locals:
```

```
C
      INTEGER*4       I
                      ! temporary variable, counter, etc
      INTEGER*2       FORM                          ! current screen id
      INTEGER*2       NUM_DEVICES                   ! number of devices to display
      INTEGER*4       STATUS                        ! function return status
      INTEGER*4       UPDATE_RATE                   ! delay between screen updates
      INTEGER*4       WAIT_TIME                     ! delay before next update?
      INTEGER*2       F_KEY                         ! function key input token
      CHARACTER*12    DEVICE( MAX_DEVICES )         ! list of devices to view
      RECORD          /DEV_BRIEF_ATT/ DEV_BRIEF_ATTRIBUTES( MAX_DEVICES )
      RECORD          /DEV_FULL_ATT/  DEV_FULL_ATTRIBUTES
      RECORD          /ITMLST/        ITEMLIST ( MAX_ITEMS + 1 )
                                      ! +1 for end of list code
      RECORD          /TOKEN/         TOKEN
                                      ! token used to represent current field
      INTEGER*2       TWO
      DATA            TWO / 2 /
      INTEGER*4       LOOP_COUNT                    ! loop counter
C
C     functions:
C
      INTEGER*4       PARSE_COMMAND_LINE            ! gets command line values
      INTEGER*4       INIT_IO                       ! defines initial I/O values
      INTEGER*4       START_IO                      ! pastes initial display to screen
      INTEGER*4       INIT_HELP                     ! defines initial help screen
      INTEGER*4       INIT_TOKEN                    ! defines initial token attributes
      INTEGER*4       INIT_COMMANDS
                      ! initialize sub-commands available
      INTEGER*4       READ_FKEY                     ! acquires all input from user
      INTEGER*4       GET_DEVICE_INFO               ! gets device information
      INTEGER*4       UPDATE_SCREEN                 ! updates the virtual display
      INTEGER*4       PROCESS_FKEY                  ! process function key from user
      INTEGER*4       HELP                          ! provides user with help
      INTEGER*4       EXECUTE_COMMAND               ! executes sub-command for user.
      INTEGER*4       UPDATE_TOKEN                  ! position token on display
      INTEGER*4       CHANGE_FORM                   ! change form display
C
C.....
C
C
C     Initialize Input/Output parameters.
C
      STATUS = INIT_IO ( )
      IF ( .NOT. STATUS ) GOTO 900                  ! error path
C
C     Parse the command line.
C
      STATUS = PARSE_COMMAND_LINE ( DEVICE(1), FORM, NUM_DEVICES,
     1                              UPDATE_RATE)
      IF ( .NOT. STATUS ) GOTO 900                  ! error path
      LAST_DEVICE(1) = NUM_DEVICES
      WAIT_TIME = UPDATE_RATE
      LOOP_COUNT = 0
```

```
C
C       Start the DISKMGR stuff.
C
        STATUS = START_IO ( FORM )
        IF ( .NOT. STATUS ) GOTO 900                    ! error path
C
C       Start a condition handler for control-y interrupts.
C
!!      STATUS = INIT_CONDITION_HANDLER ( )
        IF ( .NOT. STATUS ) GOTO 900                    ! error path
C
C       Initialize the general help form display.
C
        STATUS = INIT_HELP ( )
        IF ( .NOT. STATUS ) GOTO 900                    ! error path
C
C       Initialize the list of sub-commands available to the user.
C
        STATUS = INIT_COMMANDS ( )
C
C       Initialize token characteristics and place token on display.
C
        STATUS = INIT_TOKEN ( TOKEN, FORM )
        IF ( .NOT. STATUS ) GOTO 900                    ! error path
        STATUS = UPDATE_TOKEN ( TOKEN, F_KEY,
        1                       FORM, NUM_DEVICES )
C
C       The following do loop executes until the user presses the
C       KP0 function key or the letter E to exit.
C
        DO WHILE (( F_KEY .NE. SMG$K_TRM_KP0 ) .AND.
        1         ( F_KEY .NE. SMG$K_TRM_UPPERCASE_E ) .AND.
        2         ( F_KEY .NE. SMG$K_TRM_LOWERCASE_E ))
C
C          Check if we should erase the error message board.
C
           LOOP_COUNT = LOOP_COUNT + 1
           IF ( MOD(LOOP_COUNT, 10) .EQ. 0 ) THEN
              CALL CLEAR_ERROR
           END IF
C
C          Must update the display every UPDATE_RATE seconds.
C          On user input, must move the token to represent the
C          new current field.
C
           STATUS = READ_FKEY ( F_KEY, WAIT_TIME, FORM )
           IF ( STATUS .EQ. SS$_TIMEOUT ) THEN  ! exceeded WAIT_TIME on input
              DO I = FIRST_DEVICE(FORM), LAST_DEVICE(FORM)
                 STATUS = GET_DEVICE_INFO ( ITEMLIST, FORM, DEVICE(I),
        1                                   DEV_BRIEF_ATTRIBUTES(I),
        2                                   DEV_FULL_ATTRIBUTES )
                 IF ( .NOT. STATUS ) GOTO 900           ! error path
              END DO
              IF ( REDRAW_FLAG ) THEN
```

```
         STATUS = UPDATE_SCREEN ( FORM, NUM_DEVICES,
     1                            DEV_BRIEF_ATTRIBUTES(1),
     2                            DEV_FULL_ATTRIBUTES,
     3                            REDRAW_FLAG, TOKEN,
     4                            FIRST_DEVICE(2) )
         IF ( .NOT. STATUS ) GOTO 900          ! error path
         STATUS = UPDATE_TOKEN ( TOKEN, F_KEY,
     1                           FORM, NUM_DEVICES )
      ELSE
         STATUS = UPDATE_SCREEN ( FORM, NUM_DEVICES,
     1                            DEV_BRIEF_ATTRIBUTES(1),
     2                            DEV_FULL_ATTRIBUTES,
     3                            REDRAW_FLAG, TOKEN,
     4                            FIRST_DEVICE(2) )
         IF ( .NOT. STATUS ) GOTO 900          ! error path
      END IF
      WAIT_TIME = UPDATE_RATE

   ELSE IF ( STATUS ) THEN                     ! character was entered

      STATUS = PROCESS_FKEY ( FORM, F_KEY )
      IF ( STATUS .EQ. -1 ) THEN               ! change form requested
         STATUS = CHANGE_FORM ( FORM,
     1                          TWO,
     2                          FIRST_DEVICE(1),
     3                          LAST_DEVICE(1),
     4                          TOKEN )
         REDRAW_FLAG = .TRUE.
         STATUS = INIT_TOKEN ( TOKEN, FORM )
         STATUS = UPDATE_SCREEN ( FORM, NUM_DEVICES,
     1                            DEV_BRIEF_ATTRIBUTES(1),
     2                            DEV_FULL_ATTRIBUTES,
     3                            REDRAW_FLAG, TOKEN,
     4                            FIRST_DEVICE(2) )
         STATUS = UPDATE_TOKEN ( TOKEN, F_KEY,
     1                           FORM, NUM_DEVICES )
         WAIT_TIME = UPDATE_RATE
      ELSE IF ( STATUS .EQ. 1 ) THEN           ! update token
         STATUS = UPDATE_TOKEN ( TOKEN, F_KEY,
     1                           FORM, NUM_DEVICES )
         IF ( .NOT. STATUS ) GOTO 900          ! error path
      ELSE IF ( STATUS .EQ. 2 ) THEN           ! refresh screen
         REDRAW_FLAG = .TRUE.
         STATUS = UPDATE_SCREEN ( FORM, NUM_DEVICES,
     1                            DEV_BRIEF_ATTRIBUTES(1),
     2                            DEV_FULL_ATTRIBUTES,
     3                            REDRAW_FLAG, TOKEN,
     4                            FIRST_DEVICE(2) )
         STATUS = UPDATE_TOKEN ( TOKEN, F_KEY,
     1                           FORM, NUM_DEVICES )
         WAIT_TIME = UPDATE_RATE
      ELSE IF ( STATUS .EQ. 3 ) THEN           ! help !
         STATUS = HELP ( F_KEY, TOKEN.FIELD, FORM )
         IF ( .NOT. STATUS ) GOTO 900          ! error path
```

```
            ELSE IF ( STATUS .EQ. 4 ) THEN              ! do command
               STATUS = EXECUTE_COMMAND ( DEVICE(FIRST_DEVICE(FORM)),
     1                                    TOKEN.FIELD, FORM )
               IF ( .NOT. STATUS ) GOTO 900             ! error path
            ELSE                                        ! invalid function key: ignore
               CONTINUE
            END IF
            WAIT_TIME = WAIT_TIME / 2                   !! can we use another method???
            IF ( WAIT_TIME .LT. 1 ) THEN
               STATUS = UPDATE_SCREEN ( FORM, NUM_DEVICES,
     1                                  DEV_BRIEF_ATTRIBUTES(1),
     2                                  DEV_FULL_ATTRIBUTES,
     3                                  REDRAW_FLAG, TOKEN,
     4                                  FIRST_DEVICE(2) )
               IF ( .NOT. STATUS ) GOTO 900
               WAIT_TIME = UPDATE_RATE
            END IF

         ELSE ! .not. status

            GOTO 900

         END IF

      END DO

      GOTO 999

900   CONTINUE                          !!ERROR PATH?
      CALL LIB$PUT_SCREEN(' ', 24, 1)
      write(6,*)'DISKMGR exiting:'
      CALL LIB$SIGNAL ( %VAL(STATUS) )

999   CONTINUE
      CALL LIB$PUT_SCREEN(' ', 24, 1)
      write(6,*)
      write(6,1000) LOOP_COUNT
1000  FORMAT ( ' DISKMGR exiting successfully:  loop count = ', I10)
      END
```

A.2 CHANGE_FORM

```
      INTEGER*4 FUNCTION CHANGE_FORM ( FORM, DIM, FIRST_DEVICE,
```

1 LAST_DEVICE, TOKEN) C C name: Change form C C abstract: Sets all the needed variables to change C the current form being displayed. C C history: 01/04/88--pdl by RGD C C includes: C

```
      EXTERNAL DISKMGR_ERRUNPDIS !  error unpasting display
```

```
      INCLUDE 'MS_INC:DISK_ANAL_IO_DEFN.INC' ! I/O parameters for SMG calls
      INCLUDE 'MS_INC:TOKEN_DEFN.INC' ! token definitions C C arguments: C
      INTEGER*2 FORM ! current form being displayed
      INTEGER*2 DIM ! dimension
      INTEGER*2 FIRST_DEVICE(DIM)
      INTEGER*2 LAST_DEVICE(DIM)
      RECORD /TOKEN/ TOKEN C C globals: C
      C C locals: C
      INTEGER*4 STATUS ! function return status C C functions: C
      INTEGER*4 SMG$UNPASTE_VIRTUAL_DISPLAY ! SMG routines C C . . . . . .
. . . . . C C C Erase the current display. C
      STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(FORM), PBID )
      IF ( .NOT. STATUS ) THEN
      CALL REPORT_EXCEPTION ( 'CHANGE_FORM', STATUS,
      1 %LOC(DISKMGR__ERRUNPDIS) )
      END IF C C Swap the form values for the new form. C
      IF ( FORM .EQ. 1 ) THEN
      FORM = 2
      FIRST_DEVICE(2) = TOKEN.FIELD
      LAST_DEVICE(2) = TOKEN.FIELD
      ELSE ! form = 2
      FORM = 1
      END IF
      RETURN
      END
```

A.3 CLEAR_ERROR

```
      SUBROUTINE     CLEAR_ERROR ( )
C
C     name:          Clear Error
C
C     abstract:      Clears the error message board.
C
C     history:       03/17/88---pdl by RGD
C
C     includes:
C
      INCLUDE        '($SMGDEF)'                    ! SMG definitions
      INCLUDE        'MS_INC:DISK_ANAL_IO_DEFN.INC' ! I/O definitions
C
C     arguments:
C

C
C     globals:
C

C
C     locals:
C
      INTEGER*4      STATUS                         ! function return status
C
C     functions:
C
      INTEGER*4      SMG$PASTE_VIRTUAL_DISPLAY      ! SMG routines
      INTEGER*4      SMG$UNPASTE_VIRTUAL_DISPLAY
      INTEGER*4      SMG$ERASE_DISPLAY
C
C.....................................
C
C     Remove the current message line from the display.
C
      STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(5), PBID )
C
C     Erase the old text in the display.
C
      STATUS = SMG$ERASE_DISPLAY ( DDID(5) )
C
C     Paste the display to the pasteboard.
C
      STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(5), PBID, 24, 1 )

      RETURN
      END
```

A.4 CREATE_BRIEF_LINE

```
        INTEGER*4       FUNCTION CREATE_BRIEF_LINE ( DEV_BRIEF,
        1                                            OUTSTRING )
C
C       name:           Create Brief Line
C
C       abstract:       Creates a brief form (form=1) update line
C                       for the given device.
C
C       history:        01/07/88---pdl by RGD
C
C       includes:
C
        INCLUDE         'MS_INC:DEV_ATT.INC'
                         ! device attributes structure definition
C
C       arguments:
C
        RECORD          /DEV_BRIEF_ATT/ DEV_BRIEF
        CHARACTER*80    OUTSTRING                   ! output string
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                      ! function return status
        INTEGER*4       END                         ! end of specified substring
        REAL*4          R_MAX_BLOCKS                ! max_blocks as a real number
        REAL*4          R_FREE_BLOCKS
        REAL*4          PERCENT_USED
                        ! utilization percentage of disk space
        INTEGER*4       ERROR_LEN
                        ! length of format for error count
        DATA            ERROR_LEN       / 5 /
        INTEGER*4       MAXBLK_LEN
        DATA            MAXBLK_LEN      / 7 /
        INTEGER*4       FREEBLK_LEN
        DATA            FREEBLK_LEN     / 7 /
        INTEGER*4       PERC_LEN
        DATA            PERC_LEN        / 5 /
        INTEGER*4       TRANS_LEN
        DATA            TRANS_LEN       / 5 /
        INTEGER*4       MOUNT_LEN
        DATA            MOUNT_LEN       / 5 /
C
C       functions:
C
        INTEGER*4       INSERT_INTEGER              ! conversion routines
        INTEGER*4       INSERT_REAL
C
C.....
```

```
C
C
C       Initialize return value
C
        OUTSTRING = ' '
C
C       Insert device name ignoring leading underscore.
C
        END = MIN(13, DEV_BRIEF.DEV_NAM_LEN)
        IF ( END .GE. 2 ) THEN
           OUTSTRING( 2:END) = DEV_BRIEF.DEVICE_NAME(2:END)
        END IF
C
C       Insert volume label.
C
        END = MIN(12, DEV_BRIEF.DEV_VOL_NAME_LEN)
        IF ( END .GE. 1 ) THEN
           OUTSTRING(16:END+15) = DEV_BRIEF.VOLUME_LABEL
        END IF
C
C       Insert error count.
C
        STATUS = INSERT_INTEGER ( DEV_BRIEF.ERROR_COUNT,
        1                         OUTSTRING(30:34), ERROR_LEN )
C
C       Insert max blocks.
C
        STATUS = INSERT_INTEGER ( DEV_BRIEF.MAX_BLOCKS,
        1                         OUTSTRING(38:44), MAXBLK_LEN )
C
C       Insert free blocks.
C
        STATUS = INSERT_INTEGER ( DEV_BRIEF.FREE_BLOCKS,
        1                         OUTSTRING(48:54), FREEBLK_LEN )
C
C       Insert percent used.
C
        R_MAX_BLOCKS = DEV_BRIEF.MAX_BLOCKS
        R_FREE_BLOCKS = DEV_BRIEF.FREE_BLOCKS
        IF ( R_MAX_BLOCKS .NE. 0.0 ) THEN
           PERCENT_USED =
        1          ((R_MAX_BLOCKS-R_FREE_BLOCKS)/R_MAX_BLOCKS)*100.0
        ELSE
           PERCENT_USED = 0.0
        END IF
        STATUS = INSERT_REAL (  PERCENT_USED,
        1                       OUTSTRING(59:63), PERC_LEN, 1 )
C
C       Insert transaction count.
C
        STATUS = INSERT_INTEGER ( DEV_BRIEF.TRANS_COUNT,
        1                         OUTSTRING(68:72), TRANS_LEN )
C
C       Insert mount count.
```

```
C
      STATUS = INSERT_INTEGER ( DEV_BRIEF.MOUNT_COUNT,
      1                         OUTSTRING(76:80), MOUNT_LEN )

      RETURN
      END
```

A.5 CREATE_TOTAL_LINE

```
      INTEGER*4       FUNCTION CREATE_TOTAL_LINE ( T_ERROR_COUNT,
      1                                            T_MAX_BLOCKS,
      2                                            T_FREE_BLOCKS,
      3                                            T_TRANS_COUNT,
      4                                            T_MOUNT_COUNT,
      5                                            STRING )
C
C     name:           Create total line
C
C     abstract:       Given the input total values, produce a
C                     character string representing their values.
C
C     history:        01/08/88--pdl by RGD
C
C     includes:
C

C
C     arguments:
C
      INTEGER*4       T_ERROR_COUNT                 ! totals
      INTEGER*4       T_MAX_BLOCKS
      INTEGER*4       T_FREE_BLOCKS
      INTEGER*4       T_TRANS_COUNT
      INTEGER*4       T_MOUNT_COUNT
      CHARACTER*80    STRING
C
C     globals:
C

C
C     locals:
C
      INTEGER*4       STATUS                        ! function return status
      REAL*4          R_MAX_BLOCKS                  ! max_blocks as a real number
      REAL*4          R_FREE_BLOCKS
      REAL*4          PERCENT_USED
                      ! utilization percentage of disk space
      INTEGER*4       ERROR_LEN                     ! length of format for error count
      DATA            ERROR_LEN       / 5 /
      INTEGER*4       MAXBLK_LEN
      DATA            MAXBLK_LEN      / 7 /
      INTEGER*4       FREEBLK_LEN
```

```
      DATA            FREEBLK_LEN     / 7 /
      INTEGER*4       PERC_LEN
      DATA            PERC_LEN        / 5 /
      INTEGER*4       TRANS_LEN
      DATA            TRANS_LEN       / 5 /
      INTEGER*4       MOUNT_LEN
      DATA            MOUNT_LEN       / 5 /
C
C     functions:
C
      INTEGER*4       INSERT_INTEGER              ! conversion routines
      INTEGER*4       INSERT_REAL
C
C.....................................
C
C     Initialize return value.
C
      STRING = ' '
C
C     Insert total error count,
C
      STATUS = INSERT_INTEGER ( T_ERROR_COUNT,
     1                          STRING(1:5), ERROR_LEN )
C
C     Insert total max blocks.
C
      STATUS = INSERT_INTEGER ( T_MAX_BLOCKS,
     1                          STRING(9:15), MAXBLK_LEN )
C
C     Insert total free blocks.
C
      STATUS = INSERT_INTEGER ( T_FREE_BLOCKS,
     1                          STRING(19:25), FREEBLK_LEN )
C
C     Insert average percent used.
C
      R_MAX_BLOCKS = T_MAX_BLOCKS
      R_FREE_BLOCKS = T_FREE_BLOCKS
      PERCENT_USED=((R_MAX_BLOCKS-R_FREE_BLOCKS)/R_MAX_BLOCKS)*100.0
      STATUS = INSERT_REAL ( PERCENT_USED,
     1                       STRING(30:34), PERC_LEN, 1 )
C
C     Insert total transaction count.
C
      STATUS = INSERT_INTEGER ( T_TRANS_COUNT,
     1                          STRING(39:43), TRANS_LEN )
C
C     Insert total mount count.
C
      STATUS = INSERT_INTEGER ( T_MOUNT_COUNT,
     1                          STRING(47:51), MOUNT_LEN )

      CREATE_TOTAL_LINE = STATUS
```

RETURN
END

A.6 DEVICE_IS_DISK

C C C

LOGICAL*4 FUNCTION DEVICE_IS_DISK (DEVNAM) C C name: Device is disk? C C abstract: Determines if the given device is indeed a C disk. Returns false if disk is to valid. C C history: 02/21/88—pdl by RGD C C includes: C

EXTERNAL DISKMGR__DEVNOTEXI ! device does not exist

EXTERNAL DISKMGR__UNAVALDEV

! unable to validate device

EXTERNAL DISKMGR__DEVNOTDIS ! device is not a disk

INCLUDE '($DVIDEF)'

! GETDVI symbol definitions

INCLUDE '($DCDEF)' ! device class definitions

INCLUDE '($SSDEF)' ! GETDVI return values

INCLUDE 'MS_INC:ITMLST_STRUCT_DEFN.INC'

! itemlist structure definitions C C arguments: C

CHARACTER*(*) DEVNAM ! device name to validate C C globals: C

C C locals: C

RECORD /ITMLST/ ITEM(2) ! itemlist for GETDVI

INTEGER*4 CLASS ! returned device class

INTEGER*4 CLASS_LEN ! length of return value

INTEGER*4 STATUS ! function return status C C functions: C

INTEGER*4 SYS$GETDVIW

! retrieve device information C C . . . . C C Assume device is valid. C

```
DEVICE_IS_DISK = .TRUE.  C C Setup the itemlist description.  C
ITEM(1).BUFFER_LENGTH = 4
ITEM(1).ITEM_CODE = DVI$_DEVCLASS
ITEM(1).BUFFER_ADDRESS = %LOC(CLASS)
ITEM(1).RETURN_LENGTH = %LOC(CLASS_LEN)
ITEM(2).END_LIST = 0 C C Issue the system service call.  C
STATUS = SYS$GETDVIW ( , , %descr(DEVNAM), ITEM(1), , , , )
IF ( STATUS .EQ.  SS$_NOSUCHDEV ) THEN
DEVICE_IS_DISK = .FALSE.
CALL REPORT_EXCEPTION ( 'DEVICE_IS_DISK', STATUS,
1 %LOC(DISKMGR__DEVNOTEXI) )
ELSE IF ( .NOT.  STATUS ) THEN
CALL REPORT_EXCEPTION ( 'DEVICE_IS_DISK', SS$_DEVREQERR,
1 %LOC(DISKMGR__UNAVALDEV) )
DEVICE_IS_DISK = .FALSE.
ELSE IF ( CLASS .NE.  DC$_DISK ) THEN
CALL REPORT_EXCEPTION ( 'DEVICE_IS_DISK', STATUS,
1 %LOC(DISKMGR__DEVNOTDIS) )
DEVICE_IS_DISK = .FALSE.
END IF
RETURN
END
```

A.7 EXECUTE_COMMAND

```
      INTEGER*4       FUNCTION EXECUTE_COMMAND ( DEVICE, FIELD, FORM )
C
C     name:           Execute Command
C
C     abstract:       Execute a given command to provide the
C                     user an interface to the various VMS utilities
```

```
C                       without any knowledge of these utilities.
C
C       history:        01/19/87---pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR_ERRSAVSCR               ! error saving screen
        EXTERNAL        DISKMGR_ERRSPAWN                ! error spawing command
        EXTERNAL        DISKMGR_ERRRESSCR               ! error restoring screen
        INCLUDE         'MS_INC:FORM2_TOKEN_POS.INC'    ! max fields count
        INCLUDE         'MS_INC:COMMAND_DEFN.INC'       ! VMS utility commands
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
C
C       arguments:
C
        CHARACTER*(*)   DEVICE                          ! device to obtain info on
        INTEGER*4       FIELD                           ! current field id
        INTEGER*2       FORM                            ! current form id
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        INTEGER*4       INDEX                           ! which command to execute
        INTEGER*4       SDID                            ! saved dispaly id
!!      INTEGER*2       COMM_F_KEY                      ! input key from user
!!      INTEGER*4       COMM_WAIT_TIME
                        ! delay before returning to DISKMGR
        CHARACTER*1     A
C
C       functions:
C
!!      INTEGER*4       READ_FKEY                       ! reads any key entered
        INTEGER*4       SMG$SAVE_PHYSICAL_SCREEN        ! save screen before call
        INTEGER*4       SMG$RESTORE_PHYSICAL_SCREEN     ! restore screen after call
        INTEGER*4       LIB$SPAWN                       ! execute command and return
C
C.....
C
C
C       Define initial values for local variables.
C
!!      COMM_WAIT_TIME = 60                             ! seconds
        IF ( FORM .EQ. 1 ) THEN
           INDEX = 1
        ELSE
           INDEX = FIELD + 1
        END IF
        IF (( INDEX .EQ. 15 ) .OR.
     1    ( INDEX .EQ. 19 ) .OR.
     2    ( INDEX .EQ. 23 )) THEN
```

```
            COMMAND(INDEX)(69:80) = DEVICE(1:12)
         END IF
C
C        Since LIB$SPAWN will do I/O without using the screen management
C        facility, save the current screen image.
C
         STATUS = SMG$SAVE_PHYSICAL_SCREEN ( PBID, SDID )
         IF ( .NOT. STATUS ) THEN
            CALL REPORT_EXCEPTION ( 'EXEC_COMM', STATUS,
     1                              %LOC(DISKMGR__ERRSAVSCR) )
         END IF
C
C        Execute the desired command.
C
         STATUS = LIB$SPAWN ( COMMAND(INDEX) )
         IF ( .NOT. STATUS ) THEN
            CALL REPORT_EXCEPTION ( 'EXEC_COMM', STATUS,
     1                              %LOC(DISKMGR__ERRSPAWN) )
         END IF
C
C        Allow user to view the display.
C
         WRITE(6,10)
10       FORMAT(//,' Press Return to continue:',$)
         READ(5,20,ERR=30,END=30) A
20       FORMAT( A )
30       CONTINUE
C
C        Restore the screen to its original state.
C
         STATUS = SMG$RESTORE_PHYSICAL_SCREEN ( PBID, SDID )
         IF ( .NOT. STATUS ) THEN
            CALL REPORT_EXCEPTION ( 'EXEC_COMM', STATUS,
     1                              %LOC(DISKMGR__ERRRESSCR) )
         END IF

         EXECUTE_COMMAND = STATUS

         RETURN
         END
```

A.8 FIELD_HELP

```
         INTEGER*4       FUNCTION FIELD_HELP ( FIELD, FORM )
C
C        name:           Field Help
C
C        abstract:       Provide help for the particular field the
C                        user is currently observing.
C
C        history:        01/16/88---pdl by RGD
```

```
C
C       includes:
C
        INCLUDE         'MS_INC:FORM2_TOKEN_POS.INC'    ! max field counts
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
C
C       arguments:
C
        INTEGER*4       FIELD                           ! current field id
        INTEGER*2       FORM                            ! current form id
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        INTEGER*4       HELP_WAIT_TIME                  ! time to wait before leaving help
        INTEGER*4       HELP_WAIT_FORM                  ! help form id
        INTEGER*2       HELP_F_KEY                      ! end help function key
        INTEGER*2       INDEX                           ! array index for help text
        INTEGER*2       I                               ! temporary array index
        CHARACTER*80    HELP_TEXT(9)                    ! help text retrieved from library
C
C       functions:
C
        INTEGER*4       READ_FKEY                       ! read key to end help
        INTEGER*4       GET_FIELD_HELP                  ! retrieves field specific help
        INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY       ! SMG routines
        INTEGER*4       SMG$UNPASTE_VIRTUAL_DISPLAY
        INTEGER*4       SMG$ERASE_DISPLAY
        INTEGER*4       SMG$PUT_CHARS
C
C.....................................
C
C
C       Initialize locals.
C
        HELP_WAIT_TIME = 60                             ! seconds
        HELP_WAIT_FORM = 4
C
C       Erase the current display in the help form.
C
        DO I = 3, 9
           STATUS = SMG$ERASE_DISPLAY ( DDID(4), I, 5, I, 78 )
        END DO
C
C       Fill the display with the appropriate field help text.
C
        IF ( FORM .EQ. 1) THEN
           INDEX = 1
        ELSE
```

```
            INDEX = FIELD + 1
         END IF
C
C        Get field help text.
C
         STATUS = GET_FIELD_HELP ( HELP_TEXT, INDEX )
         IF ( .NOT. STATUS ) THEN
            FIELD_HELP = STATUS
            RETURN
         END IF
C
C        Display the field name in bold letters.
C
         STATUS = SMG$PUT_CHARS ( DDID(4),
         1                        HELP_TEXT(2)(1:25),
         2                        3, 5, , SMG$M_BOLD )
C
C        Display the field help text inside the form.
C
         DO I = 4, 8
            STATUS = SMG$PUT_CHARS( DDID(4), HELP_TEXT(I)(7:67), I+1, 10)
         END DO
C
C        Paste the help screen on the pasteboard for the user.
C
         STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(4), PBID, 13, 1 )
C
C        Wait for the user to enter any key to terminate help.
C
         STATUS = READ_FKEY( HELP_F_KEY, HELP_WAIT_TIME, HELP_WAIT_FORM)
C
C        Unpaste the help screen from the display.
C
         STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(4), PBID )

         FIELD_HELP = STATUS

         RETURN
         END
```

A.9 GENERAL_HELP

```
         INTEGER*4       FUNCTION GENERAL_HELP ( )
C
C        name:           General Help
C
C        abstract:       Provide general help on using this utility.
C                        This is a keypad layout of help similiar to
C                        what EVE displays when entering the PF2 key.
C
C        history:        01/16/88---pdl by RGD
```

```
C
C       includes:
C
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
C
C       arguments:
C

C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        INTEGER*4       HELP_WAIT_TIME                  ! time to wait before leaving help
        INTEGER*4       HELP_WAIT_FORM                  ! help form id
        INTEGER*2       HELP_F_KEY                      ! end help function key
C
C       functions:
C
        INTEGER*4       READ_FKEY                       ! read key to end help
        INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY       ! SMG routines
        INTEGER*4       SMG$UNPASTE_VIRTUAL_DISPLAY
C
C.....................................
C
C       Initialize locals.
C
        HELP_WAIT_TIME = 60                             ! seconds
        HELP_WAIT_FORM = 3
C
C       Paste the help screen on the pasteboard for the user.
C
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(3), PBID, 1, 1 )
C
C       Wait for the user to enter any key to terminate help.
C
        STATUS = READ_FKEY( HELP_F_KEY, HELP_WAIT_TIME, HELP_WAIT_FORM)
C
C       Unpaste the help screen from the display.
C
        STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(3), PBID )

        GENERAL_HELP = STATUS

        RETURN
        END
```

A.10 GET_ALL_DEVICES

```
C
C.....................................
C
      INTEGER*4       FUNCTION GET_ALL_DEVICES ( DEVICE,
      1                                          MAX_DEVICES,
      2                                          NUM_DEVICES )
C
C     name:           Get All Devices
C
C     abstract:       The /ALL qualifier is being used, therefore,
C                     we must determine all of the devices to
C                     be monitored.  This function determines
C                     all of the disks on the system.
C
C     history:        02/20/88---pdl by RGD
C
C     includes:
C
      EXTERNAL        DISKMGR_MAXDEVEXC           ! maximum devices exceeded
      EXTERNAL        SS$_OVRMAXARG               ! parameter count exceeded
C
C     arguments:
C
      INTEGER*2       MAX_DEVICES
                      ! max # devices that can be monitored
      CHARACTER*12    DEVICE ( MAX_DEVICES )      ! list of devices to monitor
      INTEGER*2       NUM_DEVICES                 ! number of devices entered
C
C     globals:
C

C
C     locals:
C
      INTEGER*4       STATUS                      ! function return status
C
C     functions:
C
      INTEGER*4       LIB$SPAWN                   ! used to find all device names
C
C.....................................
C
C
C     Spawn a DCL subprocess to issue the SHOW DEVICE D
C     command while its output is defined to be a specified file.
C
      STATUS = LIB$SPAWN ( '@MS_COM:GETDEVS' )
C
C     Open the new file for reading.
C
      OPEN ( 10, FILE='MS_COM:GETDEVS.OUT', ACCESS='SEQUENTIAL',
      1     READONLY, FORM='FORMATTED', STATUS='OLD' )
```

```
C
C       Read past the header lines.
C
        READ(10,*)
        READ(10,*)
        READ(10,*)
C
C       Begin reading the device names from the file.
C
        NUM_DEVICES = 1
        DO WHILE ( NUM_DEVICES .LE. MAX_DEVICES )
           READ( 10, 10, END=900 ) DEVICE(NUM_DEVICES)
10         FORMAT(A)
           NUM_DEVICES = NUM_DEVICES + 1
        END DO
C
C       If we fall out of the do loop, we have exceeded max_devices.
C       Notify the user of this problem.
C
        CALL REPORT_EXCEPTION ( 'GET_ALL_DEVICES', SS$_OVRMAXARG,
        1                       %LOC(DISKMGR__MAXDEVEXC) )
C
C       On termination, decrement device count to reflect the
C       true number to monitor.
C
900     NUM_DEVICES = NUM_DEVICES - 1
        GET_ALL_DEVICES = 1
        RETURN
        END
```

A.11 GET_DEV_INFO

```
        INTEGER*4 FUNCTION GET_DEVICE_INFO ( ITEMLIST, FORM, DEVICE,
        1                                    DEV_BRIEF, DEV_FULL )
C
C       name:           Get Device Information
C
C       abstract:       Get information relevant to the specified device.
C
C       history:        11/01/87--pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR__GETDVIERR              ! $GETDVI returned an error
        INCLUDE         'MS_INC:ITMLST_STRUCT_DEFN.INC' ! item list structure definition
        INCLUDE         'MS_INC:DEV_ATT.INC'
                        ! device attributes structure definition
        INCLUDE         'MS_INC:GETDVI_CONST.INC'       ! constant declarations
        INCLUDE         '($SSDEF)'                      ! system status definitions
C
```

```
C       arguments:
C
        INTEGER*2       FORM                            ! current form being displayed
        CHARACTER*12    DEVICE
                        ! device name to retrieve information
        RECORD  /DEV_BRIEF_ATT/ DEV_BRIEF               ! retrieved device specific values
        RECORD  /DEV_FULL_ATT/  DEV_FULL
        RECORD  /ITMLST/ ITEMLIST( MAX_ITEMS+1 )
                        ! add 1 for longword 0 at end of item list
C
C       globals:
C


C
C       locals:
C
!!IOSB COULD, AND SHOULD, BE A STRUCTURE TO CHECK ON RETURN FROM GETDVI...
        INTEGER*4       IOSB(2)                         ! I/O status block
        INTEGER*4       STATUS                          ! system service return code
C
C       functions:
C
        INTEGER*4       SYS$GETDVIW                     ! get system device attributes
        INTEGER*4       INIT_ITEMLIST
        INTEGER*4       SET_ITEMLIST
C
C.......................................
C
C
C       Initialize the itemlist for calls to GET_DEVICE_INFO.
C
        STATUS = INIT_ITEMLIST ( ITEMLIST, DEV_BRIEF, DEV_FULL )
        IF ( .NOT. STATUS ) THEN
           GET_DEVICE_INFO = STATUS
           RETURN
        END IF
C
C       Set the appropriate values in the itemlist
C
        STATUS = SET_ITEMLIST ( ITEMLIST, DEV_BRIEF, DEV_FULL, FORM )
        IF ( .NOT. STATUS ) THEN
           GET_DEVICE_INFO = STATUS
           RETURN
        END IF
C
C       Retrieve all needed values.
C
        STATUS = SYS$GETDVIW ( , , %DESCR(DEVICE),
        1                       ITEMLIST(1),
        2                       IOSB(1), , , )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'GETDVI', STATUS,
        1                        %LOC(DISKMGR__GETDVIERR) )
C
```

```
C          Set function return value.
C
             GET_DEVICE_INFO = STATUS
           ELSE
             IF ( FORM .EQ. 2 ) THEN
               CALL GET_PMS_DATA ( DEV_FULL )
               CALL GET_FRAG_STATS ( DEVICE, DEV_BRIEF, DEV_FULL )
             END IF
C
C          Set function return value
C
             GET_DEVICE_INFO = SS$_NORMAL
           END IF

           RETURN
           END
```

A.12 GET_FIELD_HELP

```
           INTEGER*4       FUNCTION GET_FIELD_HELP ( HELP_TEXT, INDEX )
C
C          name:           Get field help
C
C          abstract:       Get help on the field represented by INDEX.
C                          Return the help text in the HELP_TEXT character
C                          string array.
C
C          history:        02/01/88--pdl by RGD
C
C          includes:
C
           INCLUDE         '($HLPDEF)'
           INCLUDE         'MS_INC:FORM2_TOKEN_POS.INC'     ! max field count
           INCLUDE         'MS_INC:HELP_FIELD_DEFN.INC'     ! help field names
           INCLUDE         'MS_INC:LIBRARY_DEFN.INC'        ! library definitions
C
C          arguments:
C
           CHARACTER*80    HELP_TEXT(9)
           INTEGER*2       INDEX
C
C          locals:
C
           INTEGER*4       WIDTH                            ! width of return string
           INTEGER*4       FLAGS                            ! LBR flags
           INTEGER*4       STATUS                           ! function return status
           INTEGER*4       HINDEX                           ! help array index
           CHARACTER*80    TEXT(9)                          ! actual text retrieved
           INTEGER*4       I                                ! array index
C
```

```
C       functions:
C
        EXTERNAL        GET_HELP_LINE                   ! catches help text
        INTEGER*4       LBR$OUTPUT_HELP
C
C       globals:
C
        COMMON          /HELP_STUFF/ TEXT, HINDEX
C
C.....................................
C
        WIDTH  = 80                                     ! return 80 character strings
        FLAGS  = HLP$M_OUTPUT                           ! just output, no prompting
        HINDEX = 0                                      ! index of help library
        DO I = 1, 9
           TEXT(I) = ' '                                ! retrieved text goes here
        END DO
C
C       Get the text for the specified field.
C
        STATUS = LBR$OUTPUT_HELP ( GET_HELP_LINE,       ! routine to receive help text
        1                          WIDTH,               ! width of help text line
        2                          FIELD_NAME(INDEX),   ! help key to retrieve
        3                          LIBRARY_NAME,        ! library to retrieve from
        4                          FLAGS, )

        DO I = 1, 9
           HELP_TEXT(I) = TEXT(I)                       ! restore text for returning
        END DO

        GET_FIELD_HELP = STATUS                         ! set return status

        RETURN
        END
C
C
C.....................................
C
        INTEGER*4       FUNCTION GET_HELP_LINE ( OUT_LINE )
C
C       name:           Get help text line
C
C       abstract:       Retrieves the help text from the library
C
C       history:        02/01/88---pdl by RGD
C
C       includes:
C

C
C       arguments:
C
        CHARACTER*(*)   OUT_LINE                        ! the retrieved text
C
```

```
C       locals:
C
        CHARACTER*80    TEXT(9)
                        ! temporary storage of retrieved text
        INTEGER*4       HINDEX                          ! index into above array
C
C       functions:
C


C
C       globals:
C
        COMMON          /HELP_STUFF/ TEXT, HINDEX       ! locals with GET_FIELD_HELP
C
C.......................................
C
        HINDEX = HINDEX + 1                             ! increment array index
        IF ( HINDEX .LE. 9 ) THEN                       ! if within range, use help line
           TEXT ( HINDEX ) = OUT_LINE
        END IF

        GET_HELP_LINE = 1                               ! return status

        RETURN
        END
```

A.13 GET_FRAG_STATS

```
C
C.......................................
C
        SUBROUTINE      GET_FRAG_STATS ( DEVICE, DEV_BRIEF, DEV_FULL )
C
C       name:           Get Fragmentation Statistics
C
C       abstract:       Get fragmentation statistics of the specified
C                       device and return these statistics in the
C                       DEV_FULL record.
C
C       history:        03/02/88---code by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR__FRAGERROR
                        ! error opening bitmap file
        INCLUDE         'MS_INC:DEV_ATT.INC'                    ! device attributes
C
C       arguments:
C
        CHARACTER*(*)   DEVICE                                  ! device to analyze
        RECORD          /DEV_BRIEF_ATT/ DEV_BRIEF               ! retrieved values
        RECORD          /DEV_FULL_ATT/ DEV_FULL
```

```
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       MAX_FRAGS                       ! worst case # fragments
        INTEGER*4       TOTAL_FRAG                      ! total # actual fragments
        INTEGER*4       SLOT_NUM
                        ! which range current value goes into
        INTEGER*4       STATUS                          ! function return status
        INTEGER*4       SLOT_BOUNDS ( 19 )              ! slot boundaries
        DATA            SLOT_BOUNDS /     0,     3,      6,    10,
        1                              25,    50,    100,   200,
        2                             300,   500,   1000,  5000,
        3                           10000, 25000,  50000, 100000,
        4                          250000, 500000, 9999999 /
C
C       functions:
C
        INTEGER*4       SLOT                            ! returns index into usage array
        INTEGER*4       GET_FREE_BLOCKS                 ! gets # contiguos free blocks
C
C.....................................
C
C
C       Open the bitmap file for the specified device.
C
        CALL OPEN_BITMAP ( DEVICE(1:LEN(DEVICE)), STATUS )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'GET_FRAG_STATS', STATUS,
        1                          %LOC(DISKMGR__FRAGERROR) )
        ELSE
           DO SLOT_NUM = -1, 18
              DEV_FULL.FRAG_USAGE ( SLOT_NUM ) = 0
           END DO

           TOTAL_FRAG = -1

           SLOT_NUM = 0
C
C          Initialize all needed variables.
C
           MAX_FRAGS = DEV_BRIEF.FREE_BLOCKS /
        1              ( MAX ( 1,  DEV_FULL.CLUSTER_SIZE ) )
           IF ( DEV_BRIEF.FREE_BLOCKS .EQ. 0 ) RETURN
           DO WHILE ( SLOT_NUM .NE. -1 )
              SLOT_NUM = SLOT( GET_FREE_BLOCKS(), SLOT_BOUNDS, 18 )
              TOTAL_FRAG = TOTAL_FRAG + 1
              DEV_FULL.FRAG_USAGE ( SLOT_NUM ) =
        1             DEV_FULL.FRAG_USAGE ( SLOT_NUM ) + 1
           END DO
```

```
            IF ( FLOAT(MAX_FRAGS) .LT. 1 ) THEN
               DEV_FULL.FRAG_FACTOR = 0
            ELSE
               DEV_FULL.FRAG_FACTOR = ( FLOAT(TOTAL_FRAG - 1) /
     1                                  FLOAT(MAX_FRAGS) ) * 100.00
            END IF
         END IF

         CALL    CLOSE_BITMAP ( )
C
C        ***
C
         CALL    OPEN_BITMAP ( DEVICE(1:LEN(DEVICE)), STATUS )
         DO WHILE ( SLOT_NUM .NE. -1 )
            SLOT_NUM = SLOT ( GET_FREE_BLOCKS(), SLOT_BOUNDS, 18 )
         END DO
         CALL    CLOSE_BITMAP ( )

         RETURN
         END


         INTEGER*4       FUNCTION SLOT (VALUE, SLOT_BOUNDS, BOUND_COUNT)
         INTEGER*4       SLOT_BOUNDS (*)
         INTEGER*4       VALUE
         INTEGER*4       BOUND_COUNT
         INTEGER*4       I
         SLOT = -1
         DO I = 1, BOUND_COUNT
            IF ((VALUE .GT. SLOT_BOUNDS(I)) .AND.
     1      (VALUE .LE. SLOT_BOUNDS(I+1))) SLOT = I
         END DO
         RETURN
         END
```

A.14 GET_PMS_DATA

```
C
C . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
C
         SUBROUTINE      GET_PMS_DATA ( DEV_FULL )
C
C        name:           Get Performance Management Statistics Data
C
C        abstract:       Retrieves data used to describe the cumulative
C                        behavior of the file system.  The PMS data
C                        returned is found in the system executive
C                        data area in module PMSDAT.  (See VAX/VMS
C                        Internals and Data Structures, V4.4, pg 847)
C
C        history:        02/29/88---code by RGD
C
```

```
C       includes:
C
        INCLUDE         'MS_INC:DEV_ATT.INC'            ! device attribute definitions
C
C       arguments:
C
        RECORD          /DEV_FULL_ATT/ DEV_FULL         ! actual record
C
C       globals:
C
        EXTERNAL        PMS$GL_SPLIT                    ! # of split io transfers
        EXTERNAL        PMS$GL_DIRIO                    ! # direct io operations
        EXTERNAL        PMS$GL_OPENS                    ! # file opens
        EXTERNAL        PMS$GL_ERASEIO                  ! # erase $QIOs issued
        EXTERNAL        PMS$GL_TURN                     ! # window turns
        EXTERNAL        PMS$GL_HIT
                        ! # transfers not requiring window turns
        EXTERNAL        PMS$GL_DIRHIT                   ! # directory LRU hits
        EXTERNAL        PMS$GL_DIRMISS                  ! # directory LRU misses
        EXTERNAL        PMS$GL_FIDHIT                   ! # file id cache hits
        EXTERNAL        PMS$GL_FIDMISS                  ! # file id cache misses
        EXTERNAL        PMS$GL_EXTHIT                   ! # extent cache hits
        EXTERNAL        PMS$GL_EXTMISS                  ! # extent cache misses
        EXTERNAL        PMS$GL_FILHDR_HIT               ! # file header cache hits
        EXTERNAL        PMS$GL_FILHDR_MISS              ! # file header cache misses
        EXTERNAL        PMS$GL_DIRDATA_HIT              ! # directory data block hits
        EXTERNAL        PMS$GL_DIRDATA_MISS             ! # directory data block misses
        EXTERNAL        PMS$GL_STORAGMAP_HIT            ! # storage bit map cache hits
        EXTERNAL        PMS$GL_STORAGMAP_MISS           ! # storage bit map cache misses
        EXTERNAL        PMS$GL_QUOHIT                   ! # quota cache hits
        EXTERNAL        PMS$GL_QUOMISS                  ! # quota cache misses
        EXTERNAL        PMS$GL_OPEN                     ! # files currently opened
C
C       locals:
C               ( due to FORTRAN addressing, must play with externals
C                 in order to get their true values. )
C
        INTEGER*4       SPLIT                   ! # of split io transfers
        INTEGER*4       DIRIO                   ! # direct io operations
        INTEGER*4       OPENS                   ! # file opens
        INTEGER*4       ERASEIO                 ! # erase $QIOs issued
        INTEGER*4       TURN                    ! # window turns
        INTEGER*4       HIT                     ! # transfers not requiring window turns
        INTEGER*4       DIRHIT                  ! # directory LRU hits
        INTEGER*4       DIRMISS                 ! # directory LRU misses
        INTEGER*4       FIDHIT                  ! # file id cache hits
        INTEGER*4       FIDMISS                 ! # file id cache misses
        INTEGER*4       EXTHIT                  ! # extent cache hits
        INTEGER*4       EXTMISS                 ! # extent cache misses
        INTEGER*4       FILHDR_HIT              ! # file header cache hits
        INTEGER*4       FILHDR_MISS             ! # file header cache misses
        INTEGER*4       DIRDATA_HIT             ! # directory data block hits
        INTEGER*4       DIRDATA_MISS            ! # directory data block misses
        INTEGER*4       STORAGMAP_HIT           ! # storage bit map cache hits
```

```
        INTEGER*4       STORAGMAP_MISS          ! # storage bit map cache misses
        INTEGER*4       QUOHIT                  ! # quota cache hits
        INTEGER*4       QUOMISS                 ! # quota cache misses
        INTEGER*4       OPEN                    ! # files currently opened
C
C       functions:
C
        INTEGER*4       GET_VAL                 ! converts address to value
C
C.....
C
C       Get address of all PMS values.
C
        SPLIT           = %LOC( PMS$GL_SPLIT )          ! # of split io transfers
        DIRIO           = %LOC( PMS$GL_DIRIO )          ! # direct io operations
        OPENS           = %LOC( PMS$GL_OPENS )          ! # file opens
        ERASEIO         = %LOC( PMS$GL_ERASEIO )        ! # erase $QIOs issued
        TURN            = %LOC( PMS$GL_TURN )           ! # window turns
        HIT             = %LOC( PMS$GL_HIT )
                          ! # transfers not requiring window turns
        DIRHIT          = %LOC( PMS$GL_DIRHIT )         ! # directory LRU hits
        DIRMISS         = %LOC( PMS$GL_DIRMISS )        ! # directory LRU misses
        FIDHIT          = %LOC( PMS$GL_FIDHIT )         ! # file id cache hits
        FIDMISS         = %LOC( PMS$GL_FIDMISS )        ! # file id cache misses
        EXTHIT          = %LOC( PMS$GL_EXTHIT )         ! # extent cache hits
        EXTMISS         = %LOC( PMS$GL_EXTMISS )        ! # extent cache misses
        FILHDR_HIT      = %LOC( PMS$GL_FILHDR_HIT )     ! # file header cache hits
        FILHDR_MISS     = %LOC( PMS$GL_FILHDR_MISS )    ! # file header cache misses
        DIRDATA_HIT     = %LOC( PMS$GL_DIRDATA_HIT )    ! # directory data block hits
        DIRDATA_MISS    = %LOC( PMS$GL_DIRDATA_MISS )   ! # directory data block misses
        STORAGMAP_HIT   = %LOC( PMS$GL_STORAGMAP_HIT )  ! # storage bit map cache hits
        STORAGMAP_MISS  = %LOC( PMS$GL_STORAGMAP_MISS ) ! # storage bit map cache misses
        QUOHIT          = %LOC( PMS$GL_QUOHIT )         ! # quota cache hits
        QUOMISS         = %LOC( PMS$GL_QUOMISS )        ! # quota cache misses
        OPEN            = %LOC( PMS$GL_OPEN )           ! # files currently opened
C
C       Copy the value stored at these addresses to our structure.
C
        DEV_FULL.SPLIT          = GET_VAL ( %VAL(SPLIT) )
                                ! # of split io transfers
        DEV_FULL.DIRIO          = GET_VAL ( %VAL(DIRIO) )
                                ! # direct io operations
        DEV_FULL.OPENS          = GET_VAL ( %VAL(OPENS) )
                                ! # file opens
        DEV_FULL.ERASEIO        = GET_VAL( %VAL(ERASEIO) )
                                ! # erase $QIOs issued
        DEV_FULL.TURN           = GET_VAL ( %VAL(TURN) )
                                ! # window turns
        DEV_FULL.HIT            = GET_VAL ( %VAL(HIT) )
                                ! # transfers not requiring window turns
        DEV_FULL.DIRHIT         = GET_VAL ( %VAL(DIRHIT) )
                                ! # directory LRU hits
        DEV_FULL.DIRMISS        = GET_VAL ( %VAL(DIRMISS) )
                                ! # directory LRU misses
```

```
        DEV_FULL.FIDHIT          = GET_VAL ( %VAL(FIDHIT) )
                                 ! # file id cache hits
        DEV_FULL.FIDMISS         = GET_VAL ( %VAL(FIDMISS) )
                                 ! # file id cache misses
        DEV_FULL.EXTHIT          = GET_VAL ( %VAL(EXTHIT) )
                                 ! # extent cache hits
        DEV_FULL.EXTMISS         = GET_VAL ( %VAL(EXTMISS) )
                                 ! # extent cache misses
        DEV_FULL.FILHDR_HIT      = GET_VAL ( %VAL(FILHDR_HIT) )
                                 ! # file header cache hits
        DEV_FULL.FILHDR_MISS     = GET_VAL ( %VAL(FILHDR_MISS) )
                                 ! # file header cache misses
        DEV_FULL.DIRDATA_HIT     = GET_VAL ( %VAL(DIRDATA_HIT) )
                                 ! # directory data block hits
        DEV_FULL.DIRDATA_MISS    = GET_VAL ( %VAL(DIRDATA_MISS) )
                                 ! # directory data block misses
        DEV_FULL.STORAGMAP_HIT   = GET_VAL ( %VAL(STORAGMAP_HIT) )
                                 ! # storage bit map cache hits
        DEV_FULL.STORAGMAP_MISS = GET_VAL ( %VAL(STORAGMAP_MISS) )
                                 ! # storage bit map cache misses
        DEV_FULL.QUOHIT          = GET_VAL ( %VAL(QUOHIT) )
                                 ! # quota cache hits
        DEV_FULL.QUOMISS         = GET_VAL ( %VAL(QUOMISS) )
                                 ! # quota cache misses
        DEV_FULL.OPEN            = GET_VAL ( %VAL(OPEN) )
                                 ! # files currently opened

        RETURN
        END


        INTEGER*4        FUNCTION GET_VAL ( X)
        INTEGER*4        X
        GET_VAL = X
        RETURN
        END
```

A.15 GET_PROT

```
        INTEGER*4        FUNCTION GET_PROT ( PROT, STRING )
C
C       name:            Get Protection
C
C       abstract:        Given the input PROTection code, create
C                        an output STRING detailing the protection
C                        code.
C
C       history:         01/13/88---pdl by RGD
C
C       includes:
C
```

```
C
C       arguments:
C
        INTEGER*4       PROT                    ! input protection code
        CHARACTER*(*)   STRING                  ! output string
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       I                       ! class counter (SOGW)
        INTEGER*4       J                       ! access counter (RWED)
        INTEGER*4       BIT                     ! bit to test
        INTEGER*4       NEXT                    ! next positon to fill in string
C
C       functions:
C

C
C.....................................
C
C       A protection mask is expressed as a 16 bit word.
C       The first 4 bits (bits 0-3) are the system access protection.
C       Bits 4-7 are the owner access protection, bits 8-11 are the
C       group access protection, and bits 12-15 are the world access
C       protection.  Each access protection may be broken down into
C       the following for each bit.  Bit 0 is the read code, bit 1 is
C       the write code, bit 2 is the execute code, and bit 3 is the
C       delete code.  A zero in the bit position implies that access
C       is enabled while a one implies no access.
C
        STRING = ' '
        NEXT = 1
        DO I = 1, 4                             ! do for each prot code (SOGW)
           IF ( I .EQ. 1 ) THEN
              STRING(NEXT:NEXT+1) = 'S:'
           ELSE IF ( I .EQ. 2 ) THEN
              STRING(NEXT:NEXT+1) = 'O:'
           ELSE IF ( I .EQ. 3 ) THEN
              STRING(NEXT:NEXT+1) = 'G:'
           ELSE IF ( I .EQ. 4 ) THEN
              STRING(NEXT:NEXT+1) = 'W:'
           END IF
           NEXT = NEXT + 2
           DO J = 1, 4                          ! do for each access mode (RWED)
              BIT = ((I-1) * 4) + (J-1)         ! BJTEST numbers bit 0-15
              IF ( .NOT. BJTEST(PROT,BIT) ) THEN        ! is bit cleared?
C
C                Bit is clear, therefore, set protection code.
C
                 IF ( J .EQ. 1 ) THEN
                    STRING(NEXT:NEXT) = 'R'
```

```
            ELSE IF ( J .EQ. 2 ) THEN
               STRING(NEXT:NEXT) = 'W'
            ELSE IF ( J .EQ. 3 ) THEN
               STRING(NEXT:NEXT) = 'E'
            ELSE IF ( J .EQ. 4 ) THEN
               STRING(NEXT:NEXT) = 'D'
            END IF
            NEXT = NEXT + 1
         ELSE ! access not enabled
            CONTINUE
         END IF
      END DO
      STRING(NEXT:NEXT) = ','
      NEXT = NEXT + 1
   END DO
   STRING(NEXT-1:NEXT-1) = ')'

   GET_PROT = 1

   RETURN
   END
```

A.16 GET_SPECIFIED_DEVICES

```
C
C.....................................
C
      INTEGER*4       FUNCTION GET_SPECIFIED_DEVICES ( DEVICE,
      1                                                MAX_DEVICES,
      2                                                NUM_DEVICES )
C
C     name:           Get Specified Devices
C
C     abstract:       The /DEVICE qualifier is being used.
C                     Retrieve all of the devices specified on the
C                     command line.
C
C     history:        02/20/88---pdl by RGD
C
C     includes:
C
      EXTERNAL        DISKMGR__MAXDEVEXC          ! maximum devices exceeded
      INCLUDE         '($SSDEF)'                  ! system service definitions
C
C     arguments:
C
      INTEGER*2       MAX_DEVICES
                      ! max # devices that can be monitored
      CHARACTER*12    DEVICE ( MAX_DEVICES )      ! list of devices to monitor
      INTEGER*2       NUM_DEVICES                 ! number of devices entered
```

```
C
C       globals:
C

C
C       locals:
C
        INTEGER*2       CLEN                            ! length of character string
        INTEGER*4       STATUS                          ! function return status
        INTEGER*2       I                               ! temporary array index
        LOGICAL*2       FOUND
                        ! used to eliminate duplicate device strings
C
C       functions:
C
        EXTERNAL        CLI$GET_VALUE, CLI$_ABSENT      ! CLI definitions
        INTEGER*4       CLI$GET_VALUE, CLI$_ABSENT
        INTEGER*4       TRANSLATE_DEVNAM                ! logical name translation routine
        LOGICAL*4       DEVICE_IS_DISK                  ! determines if device is a disk
C
C.......................................
C
C
C       Read each of the devices in the list.
C       Qualifier has the form:  /DEVICES=(device1, device2, ... )
C
        NUM_DEVICES = 1
        DEVICE(1) = ' '
        DO WHILE ( CLI$GET_VALUE( 'DEVICES', DEVICE(NUM_DEVICES),
        1                         CLEN ) .NE. %LOC( CLI$_ABSENT ) )
C
C         Translate the device name until we get the
C         physical device name for use by GET_DEVICE_INFO.
C
          STATUS = TRANSLATE_DEVNAM ( DEVICE(NUM_DEVICES) )
          IF ( STATUS .NE. SS$_NOTRAN ) THEN
            GET_SPECIFIED_DEVICES = STATUS
            RETURN
          END IF
C
C         Eliminate duplicates.
C
          FOUND = .FALSE.
          DO I = 1, (NUM_DEVICES-1)
            IF ( DEVICE(I) .EQ. DEVICE(NUM_DEVICES) ) THEN
              FOUND = .TRUE.
            END IF
          END DO
C
C         If not a duplicate, and device is a disk, add it to our list.
C
          IF (( .NOT. FOUND) .AND.
        1     ( DEVICE_IS_DISK(DEVICE(NUM_DEVICES)) ) ) THEN
            NUM_DEVICES = NUM_DEVICES + 1
```

```
            END IF
C
C           Don't step out of bounds.
C
            IF ( NUM_DEVICES .GT. MAX_DEVICES ) THEN
               CALL REPORT_EXCEPTION ( 'GET_SPEC_DEVICES', SS$_OVRMAXARG,
     1                                 %LOC(DISKMGR__MAXDEVEXC) )
               GOTO 50
            END IF
            DEVICE(NUM_DEVICES) = ' '
         END DO
!! If terminate do loop because of too many devices, read rest of
!! argument string and produce error message saying
!! "MAX ARGS EXCEEDED, IGNORING device" or something to that effect.
!! USE MESSAGE FACILITY !!
50       NUM_DEVICES = NUM_DEVICES - 1
         GET_SPECIFIED_DEVICES = 1
         RETURN
         END
```

A.17 GET_TIME

```
         INTEGER*4       FUNCTION GET_TIME ( ATIME )
C
C        name:           Get time
C
C        abstract:
C        VAX/VMS MAINTAINS THE CURRENT DATE AND TIME IN 64-BIT FORMAT.
C        THE TIME VALUE IS A BINARY NUMBER IN 100-NANOSECOND UNITS OFFSET
C        FROM THE SYSTEM BASE DATE AND TIME, WHICH IS 00:00 0'CLOCK,
C        NOVEMBER 17, 1958 (THE SMITHSONIAN BASE DATE AND TIME FOR THE
C        ASTRONOMICAL CALENDAR).
C
C        TIME VALUES MUST BE PASSED TO, OR RETURNED FROM, SYSTEM SERVICES
C        AS THE ADDRESS OF A QUADWORD CONTAINING THE 64 BITS.
C
C
C        history:        01/01/88---pdl by RGD
C
C        includes:
C
         EXTERNAL        DISKMGR__ERRGETIME          ! error getting current time
         EXTERNAL        DISKMGR__ERRCVTIME          ! error converting time to ASCII
C
C        arguments:
C
         CHARACTER*23 ATIME
C
C        globals:
C
```

```
C
C       locals:
C
        INTEGER*4 BTIME(2)      !8 BYTES = QUADWORD
        INTEGER*2 ATIME_LEN     !STRING OF FORM "DD-MMM-YYYY HH:MM:SS.CC"
        INTEGER*4 CVTFLAG
C
C       functions:
C
        INTEGER*4 SYS$GETTIM, SYS$ASCTIM, ISTATUS
C
C.....................................
C
C
C       GET CURRENT SYSTEM TIME
C
        ISTATUS = SYS$GETTIM(BTIME(1))
        IF (.NOT. ISTATUS) THEN
           CALL REPORT_EXCEPTION ( 'GET TIME', ISTATUS,
        2                          %LOC(DISKMGR__ERRGETIME) )
           GET_TIME = ISTATUS
           RETURN
        END IF
C
C       CONVERT BINARY TIME TO ASCII STRING:
C          PARAMETERS ARE:
C          1.  LENGTH (IN BYTES) OF THE STRING TO BE RETURNED.  (OPTIONAL)
C          2.  BUFFER FOR STORING THE ASCII STRING RETURNED.  (REQUIRED)
C          3.  TIME VALUE THAT IS TO BE CONVERTED.  (OPTIONAL)
C          4.  CONVERT FLAG  (0 ==> RETURN DATE AND TIME,
C                             1 ==> RETURN ONLY THE TIME.)  (OPTIONAL)
C                            (PASSED BY VALUE, NOT BY REFERENCE)
C
        ATIME_LEN = 23
        CVTFLAG = 0
        ISTATUS = SYS$ASCTIM(ATIME_LEN, ATIME, BTIME(1), %VAL(CVTFLAG))
        IF (.NOT. ISTATUS) THEN
           CALL REPORT_EXCEPTION ( 'GET TIME', ISTATUS,
        1                          %LOC(DISKMGR__ERRCVTIME) )
           GET_TIME = ISTATUS
           RETURN
        END IF
        GET_TIME = ISTATUS
        RETURN
        END
```

A.18 GET_TOTALS

```
        SUBROUTINE      GET_TOTALS ( T_ERROR_COUNT, T_MAX_BLOCKS,
        1                            T_FREE_BLOCKS, T_TRANS_COUNT,
        2                            T_MOUNT_COUNT, DEV_BRIEF,
```

```
      3                         NUM_DEVICES )
C
C     name:           Get Totals
C
C     abstract:       Get the totals for all NUM_DEVICES devices.
C
C     history:        01/08/88--pdl by RGD
C
C     includes:
C
      INCLUDE         'MS_INC:DISK_ANAL_CONST.INC'    ! constant definitions
      INCLUDE         'MS_INC:DEV_ATT.INC'
                      ! device attributes structure definition
C
C     arguments:
C
      INTEGER*4       T_ERROR_COUNT                   ! totals
      INTEGER*4       T_MAX_BLOCKS
      INTEGER*4       T_FREE_BLOCKS
      INTEGER*4       T_TRANS_COUNT
      INTEGER*4       T_MOUNT_COUNT
      RECORD          /DEV_BRIEF_ATT/ DEV_BRIEF ( MAX_DEVICES )
      INTEGER*2       NUM_DEVICES
C
C     globals:
C


C
C     locals:
C
      INTEGER*4       I                               ! index/counter
C
C     functions:
C


C
C.....................................
C
C     Initialize totals.
C
      T_ERROR_COUNT = 0
      T_MAX_BLOCKS  = 0
      T_FREE_BLOCKS = 0
      T_TRANS_COUNT = 0
      T_MOUNT_COUNT = 0
C
C     Determine totals for all the devices.
C
      DO I = 1, NUM_DEVICES
         T_ERROR_COUNT = T_ERROR_COUNT + DEV_BRIEF(I).ERROR_COUNT
         T_MAX_BLOCKS = T_MAX_BLOCKS + DEV_BRIEF(I).MAX_BLOCKS
         T_FREE_BLOCKS = T_FREE_BLOCKS + DEV_BRIEF(I).FREE_BLOCKS
         T_TRANS_COUNT = T_TRANS_COUNT + DEV_BRIEF(I).TRANS_COUNT
         T_MOUNT_COUNT = T_MOUNT_COUNT + DEV_BRIEF(I).MOUNT_COUNT
```

```
        END DO

        RETURN
        END
```

A.19 HELP

```
        INTEGER*4       FUNCTION HELP ( F_KEY, FIELD, FORM )
C
C       name:           HELP!
C
C       abstract:       Provide the user with help.  There are two
C                       types of help available for this utility.
C                       The first type of help is a general help
C                       which shows the keypad layout and other
C                       pre-defined keys for using the tool.  This
C                       help is obtained when entering the PF2 key.
C                       The second type of help is a more specific
C                       help and is based on the current field
C                       the user is in.  This help is obtained by
C                       entering the PF3 key.
C
C       history:        01/16/88-pdl by RGD
C
C       includes:
C
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
C
C       arguments:
C
        INTEGER*2       F_KEY                           ! function key entered by user
        INTEGER*4       FIELD                           ! current field id
        INTEGER*2       FORM                            ! current screen id
C
C       globals:
C


C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
C
C       functions:
C
        INTEGER*4       GENERAL_HELP                    ! general help for keypad
        INTEGER*4       FIELD_HELP                      ! specific help for current field
C
C.......................................
C
        IF ( F_KEY .EQ. SMG$K_TRM_PF2 ) THEN            ! general help
           STATUS = GENERAL_HELP ( )
```

```
      ELSE IF ( F_KEY .EQ. SMG$K_TRM_PF3 ) THEN       ! field help
        STATUS = FIELD_HELP ( FIELD, FORM )
      END IF

      HELP = 1

      RETURN
      END
```

A.20 INIT_COMMANDS

```
      INTEGER*4       FUNCTION INIT_COMMANDS ( )
C
C     name:           Initialize commands
C
C     abstract:       Initializes all sub-commands available for
C                     use by this utility.  In general, each field
C                     has a command associated with it that can be
C                     used to generate more information relating
C                     to the field.
C
C     history:        01/19/87--pdl by RGD
C
C     includes:
C
      INCLUDE         'MS_INC:FORM2_TOKEN_POS.INC'    ! max fields count
      INCLUDE         'MS_INC:COMMAND_DEFN.INC'       ! command definitions
C
C     arguments:
C

C
C     globals:
C

C
C     locals:
C
      INTEGER*4       I                               ! scratch variable
C
C     functions:
C

C
C.....................................
C
C
C     Initialize the command array.
C
      DO I = 1, 24
        COMMAND(I) = 'SHOW DEV D'
```

```
        END DO

        COMMAND(15) = 'SHOW DEV/FILES'
        COMMAND(19) = 'SHOW DEV/FILES'
        COMMAND(23) = 'ANALYZE/ERROR/SUMMARY/INCLUDE='
        COMMAND(25) = 'MONITOR DISK/ITEM=ALL'
        COMMAND(26) = 'MONITOR PROCESS/TOPDIO'

        DO I = 27, 29
           COMMAND(I) = 'MONITOR FCP'
        END DO

        DO I = 30, MAX_FIELDS
           COMMAND(I) = 'MONITOR FILE_SYSTEM_CACHE'
        END DO

        COMMAND(33) = 'MONITOR FCP'
        COMMAND(45) = 'MONITOR FCP'
        COMMAND(46) = 'SHOW DEV D'
        COMMAND(47) = 'SHOW DEV D'


        INIT_COMMANDS = 1

        RETURN
        END
```

A.21 INIT_HELP

```
        INTEGER*4       FUNCTION INIT_HELP ( )
C
C       name:           Initialize help
C
C       abstract:       Initialize the general help screen.  This
C                       display is constant and is not ever modified
C                       by this program.  Also, this routine initializes
C                       the structure of the field help screen and
C                       opens the DISKMGR specific help library for
C                       read access.
C
C       history:        01/16/88--pdl by RGD
C
C       includes:
C
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
        INCLUDE         'MS_INC:LIBRARY_DEFN.INC'       ! library definitions
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
        INCLUDE         '($LBRDEF)'                     ! library defintions
        INCLUDE         '($HLPDEF)'                     ! help constants
C
```

```
C       arguments:
C
C
C
C       globals:
C
C
C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
C
C       functions:
C
        INTEGER*4       SMG$BEGIN_DISPLAY_UPDATE        ! SMG routines
        INTEGER*4       SMG$DRAW_RECTANGLE
        INTEGER*4       SMG$DRAW_LINE
        INTEGER*4       SMG$PUT_CHARS
        INTEGER*4       SMG$END_DISPLAY_UPDATE
        INTEGER*4       LER$INI_CONTROL                 ! library routines
        INTEGER*4       LER$OPEN
C
C.....................................
C
        STATUS = SMG$BEGIN_DISPLAY_UPDATE ( DDID(3) )
C
C       Draw arrow key boxes.
C
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1,  2,  6, 11 ) ! up arrow
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 11,  6, 20 ) ! down arrow
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 20,  6, 29 ) ! left arrow
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 29,  6, 38 ) ! right arrow
C
C       Draw keypad boxes.
C
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 44,  5, 53 ) ! PF1
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 53,  5, 62 ) ! PF2
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 62,  5, 71 ) ! PF3
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  1, 71,  5, 80 ) ! PF4

        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  5, 44,  9, 53 ) ! KP7
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  5, 53,  9, 62 ) ! KP8
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  5, 62,  9, 71 ) ! KP9
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  5, 71,  9, 80 ) ! KP-

        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  9, 44, 13, 53 ) ! KP4
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  9, 53, 13, 62 ) ! KP5
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  9, 62, 13, 71 ) ! KP6
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3),  9, 71, 13, 80 ) ! KP+

        STATUS = SMG$DRAW_RECTANGLE ( DDID(3), 13, 44, 17, 53 ) ! KP1
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3), 13, 53, 17, 62 ) ! KP2
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3), 13, 62, 17, 71 ) ! KP3
        STATUS = SMG$DRAW_RECTANGLE ( DDID(3), 13, 71, 21, 80 ) ! KPenter
```

```
      STATUS = SMG$DRAW_RECTANGLE ( DDID(3), 17, 44, 21, 62 ) ! KP0
      STATUS = SMG$DRAW_RECTANGLE ( DDID(3), 17, 62, 21, 71 ) ! KP.
C
C     Insert text into the boxes.
C
      STATUS = SMG$PUT_CHARS ( DDID(3), '^',       2, 5 )     ! up arrow
      STATUS = SMG$DRAW_LINE ( DDID(3), 3, 5, 4, 5 )
      STATUS = SMG$PUT_CHARS ( DDID(3), 'UP',      5, 5 )

      STATUS = SMG$DRAW_LINE ( DDID(3), 2, 14, 3, 14 )        ! down arrow
      STATUS = SMG$PUT_CHARS ( DDID(3), 'v',       4, 14 )
      STATUS = SMG$PUT_CHARS ( DDID(3), 'DOWN',    5, 13 )

      STATUS = SMG$PUT_CHARS ( DDID(3), '<',       4, 22 )    ! left arrow
      STATUS = SMG$DRAW_LINE ( DDID(3), 4, 23, 4, 25 )
      STATUS = SMG$PUT_CHARS ( DDID(3), 'LEFT',    5, 22 )

      STATUS = SMG$DRAW_LINE ( DDID(3), 4, 31, 4, 33 )        ! right arrow
      STATUS = SMG$PUT_CHARS ( DDID(3), '>',       4, 34 )
      STATUS = SMG$PUT_CHARS ( DDID(3), 'RIGHT',   5, 31 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'DISKMGR', 3, 54 )    ! PF2
      STATUS = SMG$PUT_CHARS ( DDID(3), 'HELP',    4, 55 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'FIELD',   3, 64 )    ! PF3
      STATUS = SMG$PUT_CHARS ( DDID(3), 'HELP',    4, 64 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'DO',      3, 74 )    ! PF4
      STATUS = SMG$PUT_CHARS ( DDID(3), 'COMMAND', 4, 72 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'REFRESH', 7, 45 )    ! KP7

      STATUS = SMG$PUT_CHARS ( DDID(3), 'PREV',    7, 55 )    ! KP8
      STATUS = SMG$PUT_CHARS ( DDID(3), 'FIELD',   8, 55 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'PREV',   11, 46 )    ! KP4
      STATUS = SMG$PUT_CHARS ( DDID(3), 'FIELD',  12, 46 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'NEXT',   11, 64 )    ! KP6
      STATUS = SMG$PUT_CHARS ( DDID(3), 'FIELD',  12, 64 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'NEXT',   15, 55 )    ! KP2
      STATUS = SMG$PUT_CHARS ( DDID(3), 'FIELD',  16, 55 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'CHANGE', 16, 72 )    ! KPenter
      STATUS = SMG$PUT_CHARS ( DDID(3), 'FORM',   17, 73 )

      STATUS = SMG$PUT_CHARS ( DDID(3), 'EXIT',   19, 46 )    ! KP0
C
C     Insert information on useful non-keypad keys.
C
      STATUS = SMG$PUT_CHARS(DDID(3),'BACKSPACE  Previous field', 8,1)
      STATUS = SMG$PUT_CHARS(DDID(3),'TAB        Next field',     9,1)
```

```
        STATUS = SMG$PUT_CHARS(DDID(3),'<CR>      Change form',   10,1)
        STATUS = SMG$PUT_CHARS(DDID(3),'CTRL/H    Previous field',11,1)
        STATUS = SMG$PUT_CHARS(DDID(3),'CTRL/I    Next field',    12,1)
        STATUS = SMG$PUT_CHARS(DDID(3),'CTRL/R    Refresh screen',13,1)
        STATUS = SMG$PUT_CHARS(DDID(3),'CTRL/W    Refresh screen',14,1)
C
C       Output message describing how to terminate help display.
C
        STATUS = SMG$PUT_CHARS ( DDID(3),
     1 'Press any key to continue
     2                  ',
     3 23, 1, , SMG$M_REVERSE )

        STATUS = SMG$END_DISPLAY_UPDATE ( DDID(3) )
C
C       Initialize the field specific help screen.  NOTE:  this
C       display changes based on the current field, therefore,
C       we can only initialize the background portion of the display.
C
        STATUS = SMG$BEGIN_DISPLAY_UPDATE ( DDID(4) )
        STATUS = SMG$DRAW_RECTANGLE ( DDID(4), 1, 1, 12, 80 )
        STATUS = SMG$DRAW_RECTANGLE ( DDID(4), 2, 2, 10, 79 )
        STATUS = SMG$PUT_CHARS ( DDID(4),
     1 'Press any key to continue
     2                ',
     3 11, 3, , SMG$M_REVERSE )
        STATUS = SMG$END_DISPLAY_UPDATE ( DDID(4) )
C
C       Initialize the necessary function codes for the LBR routines.
C
        FUNCTION = LBR$C_READ                   ! read function code
        TYPE     = LBR$C_TYP_HLP                ! help text library
C
C       Initialize control of the library.
C
        STATUS = LBR$INI_CONTROL ( LIBRARY_INDEX,
     1                             FUNCTION,
     2                             TYPE )
        IF ( .NOT. STATUS ) THEN
           INIT_HELP = STATUS
           RETURN
        END IF
C
C       Open the DISKMGR help library.
C
        STATUS = LBR$OPEN ( LIBRARY_INDEX,
     1                      LIBRARY_NAME,
     2                      ,,, )
        IF ( .NOT. STATUS ) THEN
           INIT_HELP = STATUS
           RETURN
        END IF

        INIT_HELP = STATUS
```

```
        RETURN
        END
```

A.22 INIT_IO

```
        INTEGER*4       FUNCTION INIT_IO ( )
C
C       name:           Initialize Input/Output
C
C       abstract:       Create needed pasteboards, virtual displays,
C                       and virtual keyboards.
C
C       history:        11/01/87---pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR__ERRCREPBD              ! error creating pasteboard
        EXTERNAL        DISKMGR__ERRCREDIS              ! error creating virtual display
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! global IO definitions
        INCLUDE         'MS_INC:DISK_ANAL_CONST.INC'    ! constant definitions
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
C
C       arguments:
C

C
C       globals:
C
C
C       locals:
C
        INTEGER*4       STATUS                          ! return status
C
C       functions:
C
        INTEGER*4       SMG$CREATE_PASTEBOARD           ! screen management routines
        INTEGER*4       SMG$CREATE_VIRTUAL_DISPLAY
        INTEGER*4       SMG$CREATE_VIRTUAL_KEYBOARD
        INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY
C
C.....................................
C
C
C       Create a pasteboard associated with the physical terminal
C       the user is using.  Screen management facility of VMS
C       provides all the necessary operations for displaying the
C       information on the physical device.
C
C
C       Create virtual display for error messages.
```

```
C
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY ( 1,        ! number of rows
     1                                      132,      ! number of columns
     2                                      DDID(5) ) ! field help display id
      IF ( .NOT. STATUS ) THEN
         WRITE(6,*)'DISKMGR:  Error create ERROR virtual display.'
         INIT_IO = STATUS
         RETURN
      END IF
C
C     Create the pasteboard.
C
      STATUS = SMG$CREATE_PASTEBOARD ( PBID,          ! pasteboard identifier
     1                                 'SYS$OUTPUT',  ! associated output device
     2                                 ROWS,          ! number rows of physical device
     3                                 COLS )
                                      ! number columns of physical device
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INIT_IO', STATUS,
     1                           %LOC(DISKMGR__ERRCREPBD) )
         INIT_IO = STATUS
         RETURN
      END IF
C
C     Create the needed virtual displays.  There are two displays
C     that must be created ( one for each form ).  This will allow
C     easy recovery when PREVIOUS_SCREEN function key is entered.
C
C
C     Create data virtual display for brief form.
C
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY ( 23,       ! number of rows
     1                                      COLS,     ! number of columns
     2                                      DDID(1) ) ! data display id
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INIT_IO', STATUS,
     1                           %LOC(DISKMGR__ERRCREDIS) )
         INIT_IO = STATUS
         RETURN
      END IF
C
C     Create data virtual display for full form.
C
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY ( 23,       ! number of rows
     1                                      COLS,     ! number of columns
     2                                      DDID(2) ) ! data display id
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INIT_IO', STATUS,
     1                           %LOC(DISKMGR__ERRCREDIS) )
         INIT_IO = STATUS
         RETURN
      END IF
C
C     Create data virtual display for the help forms.
```

```
C
      STATUS = SMG$CREATE_VIRTUAL_DISPLAY ( 24,       ! number of rows
      1                                     COLS,     ! number of columns
      2                                     DDID(3) ) ! general help display id
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INIT_IO', STATUS,
      1                          %LOC(DISKMGR__ERRCREDIS) )
         INIT_IO = STATUS
         RETURN
      END IF

      STATUS = SMG$CREATE_VIRTUAL_DISPLAY ( 12,       ! number of rows
      1                                     COLS,     ! number of columns
      2                                     DDID(4) ) ! field help display id
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INIT_IO', STATUS,
      1                          %LOC(DISKMGR__ERRCREDIS) )
         INIT_IO = STATUS
         RETURN
      END IF
C
C     Create virtual keyboard for device independent input operations
C
      STATUS = SMG$CREATE_VIRTUAL_KEYBOARD ( VKID, 'SYS$OUTPUT' )
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INIT_IO', STATUS,
      1                          %LOC(DISKMGR__ERRCREDIS) )
         INIT_IO = STATUS
         RETURN
      END IF

      INIT_IO = STATUS
      RETURN
      END
```

A.23 INIT_ITEMLIST

```
      INTEGER*4       FUNCTION INIT_ITEMLIST ( ITEMLIST , DEV_BRIEF,
      1                                        DEV_FULL )
C
C     name:           Initialize itemlist
C
C     abstract:       Initialize the itemlist structure for
C                     GET_DEVICE_INFO function.
C
C     history:        01/01/88--pdl by RGD
C
C     includes:
C
      INCLUDE         'MS_INC:DEV_ATT.INC'
                      ! device attributes structure definition
```

```
      INCLUDE         'MS_INC:ITMLST_STRUCT_DEFN.INC' ! itemlist structure definitions
      INCLUDE         '($DVIDEF)'                     ! GETDVI symbol definitions
C
C     arguments:
C
      RECORD  /ITMLST/        ITEMLIST ( MAX_ITEMS + 1) ! +1 for end of list code
      RECORD  /DEV_BRIEF_ATT/ DEV_BRIEF
      RECORD  /DEV_FULL_ATT/  DEV_FULL
C
C     globals:
C


C
C     locals:
C
      INTEGER*4       DUMMY                           ! dummy return value
C
C     functions:
C
      ITEMLIST( 1).BUFFER_LENGTH  = 64
      ITEMLIST( 1).ITEM_CODE      = DVI$_DEVNAM
      ITEMLIST( 1).BUFFER_ADDRESS = %LOC(DEV_BRIEF.DEVICE_NAME)
      ITEMLIST( 1).RETURN_LENGTH  = %LOC(DEV_BRIEF.DEV_NAM_LEN)

      ITEMLIST( 2).BUFFER_LENGTH  = 12
      ITEMLIST( 2).ITEM_CODE      = DVI$_VOLNAM
      ITEMLIST( 2).BUFFER_ADDRESS = %LOC(DEV_BRIEF.VOLUME_LABEL)
      ITEMLIST( 2).RETURN_LENGTH  = %LOC(DEV_BRIEF.DEV_VOL_NAME_LEN)

      ITEMLIST( 3).BUFFER_LENGTH  = 4
      ITEMLIST( 3).ITEM_CODE      = DVI$_ERRCNT
      ITEMLIST( 3).BUFFER_ADDRESS = %LOC(DEV_BRIEF.ERROR_COUNT)
      ITEMLIST( 3).RETURN_LENGTH  = %LOC(DUMMY)

      ITEMLIST( 4).BUFFER_LENGTH  = 4
      ITEMLIST( 4).ITEM_CODE      = DVI$_MAXBLOCK
      ITEMLIST( 4).BUFFER_ADDRESS = %LOC(DEV_BRIEF.MAX_BLOCKS)
      ITEMLIST( 4).RETURN_LENGTH  = %LOC(DUMMY)

      ITEMLIST( 5).BUFFER_LENGTH  = 4
      ITEMLIST( 5).ITEM_CODE      = DVI$_FREEBLOCKS
      ITEMLIST( 5).BUFFER_ADDRESS = %LOC(DEV_BRIEF.FREE_BLOCKS)
      ITEMLIST( 5).RETURN_LENGTH  = %LOC(DUMMY)

      ITEMLIST( 6).BUFFER_LENGTH  = 4
      ITEMLIST( 6).ITEM_CODE      = DVI$_TRANSCNT
      ITEMLIST( 6).BUFFER_ADDRESS = %LOC(DEV_BRIEF.TRANS_COUNT)
      ITEMLIST( 6).RETURN_LENGTH  = %LOC(DUMMY)

      ITEMLIST( 7).BUFFER_LENGTH  = 4
      ITEMLIST( 7).ITEM_CODE      = DVI$_MOUNTCNT
      ITEMLIST( 7).BUFFER_ADDRESS = %LOC(DEV_BRIEF.MOUNT_COUNT)
      ITEMLIST( 7).RETURN_LENGTH  = %LOC(DUMMY)
```

```
ITEMLIST( 8).BUFFER_LENGTH  = 10
ITEMLIST( 8).ITEM_CODE      = DVI$_MEDIA_NAME
ITEMLIST( 8).BUFFER_ADDRESS = %LOC(DEV_FULL.MEDIA_NAME)
ITEMLIST( 8).RETURN_LENGTH  = %LOC(DEV_FULL.MEDNAM_LEN)

ITEMLIST( 9).BUFFER_LENGTH  = 4
ITEMLIST( 9).ITEM_CODE      = DVI$_MAXFILES
ITEMLIST( 9).BUFFER_ADDRESS = %LOC(DEV_FULL.MAX_FILES)
ITEMLIST( 9).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(10).BUFFER_LENGTH  = 4
ITEMLIST(10).ITEM_CODE      = DVI$_ACPTYPE
ITEMLIST(10).BUFFER_ADDRESS = %LOC(DEV_FULL.ACP_TYPE)
ITEMLIST(10).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(11).BUFFER_LENGTH  = 4
ITEMLIST(11).ITEM_CODE      = DVI$_MNT
ITEMLIST(11).BUFFER_ADDRESS = %LOC(DEV_FULL.MOUNTED)
ITEMLIST(11).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(12).BUFFER_LENGTH  = 4
ITEMLIST(12).ITEM_CODE      = DVI$_PID
ITEMLIST(12).BUFFER_ADDRESS = %LOC(DEV_FULL.PID)
ITEMLIST(12).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(13).BUFFER_LENGTH  = 4
ITEMLIST(13).ITEM_CODE      = DVI$_OWNUIC
ITEMLIST(13).BUFFER_ADDRESS = %LOC(DEV_FULL.OWNUIC.UIC)
ITEMLIST(13).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(14).BUFFER_LENGTH  = 4
ITEMLIST(14).ITEM_CODE      = DVI$_VPROT
ITEMLIST(14).BUFFER_ADDRESS = %LOC(DEV_FULL.PROT)
ITEMLIST(14).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(15).BUFFER_LENGTH  = 4
ITEMLIST(15).ITEM_CODE      = DVI$_CLUSTER
ITEMLIST(15).BUFFER_ADDRESS = %LOC(DEV_FULL.CLUSTER_SIZE)
ITEMLIST(15).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(16).BUFFER_LENGTH  = 4
ITEMLIST(16).ITEM_CODE      = DVI$_CYLINDERS
ITEMLIST(16).BUFFER_ADDRESS = %LOC(DEV_FULL.NUM_CYLINDERS)
ITEMLIST(16).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(17).BUFFER_LENGTH  = 4
ITEMLIST(17).ITEM_CODE      = DVI$_TRACKS
ITEMLIST(17).BUFFER_ADDRESS = %LOC(DEV_FULL.NUM_TRACKS)
ITEMLIST(17).RETURN_LENGTH  = %LOC(DUMMY)

ITEMLIST(18).BUFFER_LENGTH  = 4
ITEMLIST(18).ITEM_CODE      = DVI$_SECTORS
ITEMLIST(18).BUFFER_ADDRESS = %LOC(DEV_FULL.NUM_SECTORS)
ITEMLIST(18).RETURN_LENGTH  = %LOC(DUMMY)
```

```
        ITEMLIST(19).BUFFER_LENGTH  = 4
        ITEMLIST(19).ITEM_CODE      = DVI$_OPCNT
        ITEMLIST(19).BUFFER_ADDRESS = %LOC(DEV_FULL.OPS_COMPLETE)
        ITEMLIST(19).RETURN_LENGTH  = %LOC(DUMMY)

        ITEMLIST(20).BUFFER_LENGTH  = 4
        ITEMLIST(20).ITEM_CODE      = DVI$_DEVBUFSIZ
        ITEMLIST(20).BUFFER_ADDRESS = %LOC(DEV_FULL.DEF_BUFF_SZ)
        ITEMLIST(20).RETURN_LENGTH  = %LOC(DUMMY)

        ITEMLIST(21).BUFFER_LENGTH  = 4
        ITEMLIST(21).ITEM_CODE      = DVI$_REFCNT
        ITEMLIST(21).BUFFER_ADDRESS = %LOC(DEV_FULL.REF_COUNT)
        ITEMLIST(21).RETURN_LENGTH  = %LOC(DUMMY)

        ITEMLIST ( MAX_ITEMS + 1).END_LIST = 0

        INIT_ITEMLIST = 1

        RETURN
        END
```

A.24 INIT_TOKEN

```
        INTEGER*4       FUNCTION INIT_TOKEN ( TOKEN, FORM )
C
C       name:           Initialize token
C
C       abstract:       Initialize token attributes.
C
C       history:        01/03/88--pdl by RGD
C
C       includes:
C
        INCLUDE         'MS_INC:TOKEN_DEFN.INC'         ! token definitions
C
C       arguments:
C
        RECORD          /TOKEN/ TOKEN                   ! output token
        INTEGER*2       FORM                            ! current screen id
C
C       globals:
C

C
C       locals:
C

C
C       functions:
```

```
C
C
C.....
C
C
C       Set token to default initial values.
C
        IF ( FORM .EQ. 1 ) THEN
           TOKEN.ROW = 7
           TOKEN.COL = 1
        ELSE ! assume form = 2
           TOKEN.ROW = 4
           TOKEN.COL = 1
        END IF

        TOKEN.FIELD = 1
        TOKEN.CHR = '*'

        INIT_TOKEN = 1

        RETURN
        END
```

A.25 INSERT_INTEGER

```
        INTEGER*4       FUNCTION INSERT_INTEGER ( VALUE, STRING, LENG )
C
C       name:           Insert Integer
C
C       abstract:       Insert the integer VALUE into STRING using the
C                       fortran format of I<LENG>.
C
C       history:        01/07/88--pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR_ERRCVTVAL               ! error converting value
C
C       arguments:
C
        INTEGER*4       VALUE
        CHARACTER*(*)   STRING
        INTEGER*4       LENG
C
C       globals:
C

C
C       locals:
C
```

```
      INTEGER*4       STATUS                      ! function return status
      INTEGER*4       SLEN                        ! string lenght of STRING
      INTEGER*4       I                           ! index/counter
      LOGICAL         REMOVE_ZEROES               ! remove leading zeroes?
C
C     functions:
C
      INTEGER*4       OTS$CVT_L_TI                ! conversion routines
C
C.....
C
C     Determine if we should remove leading zeroes in output string.
C     If the input length argument is greater than zero, then we
C     should remove leading zeroes.
C
      IF ( LENG .GT. 0 ) THEN
         REMOVE_ZEROES = .TRUE.
      ELSE
         REMOVE_ZEROES = .FALSE.
         LENG = LENG * -1
      END IF
C
C     Convert the integer value to a character string.
C
      STATUS = OTS$CVT_L_TI ( VALUE,
     1                        %DESCR(STRING),
     2                        %VAL(LENG),, )
      IF ( .NOT. STATUS ) THEN
         CALL REPORT_EXCEPTION ( 'INSERT_INTEGER', STATUS,
     1                           %LOC(DISKMGR__ERRCVTVAL) )
         STRING(1:1) = '*'
         INSERT_INTEGER = STATUS
         RETURN
      END IF
C
C     Remove leading zeroes if desired.
C
      IF ( REMOVE_ZEROES ) THEN
         SLEN = LEN ( STRING )
         I = 1
         DO WHILE (( STRING(I:I) .EQ. '0' ) .AND. ( I .LT. SLEN ))
            STRING(I:I) = ' '
            I = I + 1
         END DO
      END IF

      INSERT_INTEGER = 1

      RETURN
      END
```

A.26 INSERT_REAL

```
      INTEGER*4       FUNCTION INSERT_REAL ( VALUE, STRING,
      1                                      LENG,  RESOLUTION )
C
C     name:           Insert Real
C
C     abstract:       Insert the real VALUE into STRING using the
C                     fortran format of F<LENG>.RESOLUTION.
C
C     history:        01/07/88--pdl by RGD
C
C     includes:
C

C
C     arguments:
C
      REAL*4          VALUE
      CHARACTER*(*)   STRING
      INTEGER*4       LENG
      INTEGER*4       RESOLUTION
C
C     globals:
C

C
C     locals:
C

C
C     functions:
C

C
C.....................................
C
C     Convert the real value to a character string.
C
      WRITE(STRING(1:LENG), 10) VALUE
10    FORMAT(F<LENG>.<RESOLUTION>)

      INSERT_REAL = 1   !! assume it will never fail for now...

      RETURN
      END
```

A.27 OUTPUT_CHARS

```
      INTEGER*4       FUNCTION OUTPUT_CHARS ( STRING, LENG, ROW, COL )
C
```

```
C       name:           Output Characters
C
C       abstract:       Output the characters STRING of length LENG
C                       on the display at cursor position ROW, COL.
C
C       history:        01/13/88--pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR_ERRPUTCHR
                        ! error writing characters to display
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
C
C       arguments:
C
        CHARACTER*(*)   STRING                          ! string to be displayed
        INTEGER*4       LENG                            ! length of output string
        INTEGER*4       ROW                             ! physical  row of display
        INTEGER*4       COL                             ! physical column
C
C       globals:
C


C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
C
C       functions:
C
        INTEGER*4       SMG$PUT_CHARS                   ! SMG routine
C
C.......................................
C
        STATUS = 1
        IF ( LENG .GE. 1 ) THEN
           STATUS = SMG$PUT_CHARS ( DDID(2), STRING(1:LENG), ROW, COL )
           IF ( .NOT. STATUS ) THEN
              CALL REPORT_EXCEPTION ( 'OUTPUT_CHARS', STATUS,
     1                                %LOC(DISKMGR_ERRPUTCHR))
           END IF
        END IF

        OUTPUT_CHARS = STATUS

        RETURN
        END
```

A.28 OUTPUT_INTEGER

```
      INTEGER*4       FUNCTION OUTPUT_INTEGER ( VALUE, ROW,
     1                                          COL, LENG )
C
C     name:           Output integer.
C
C     abstract:       Output an integer value to the screen.
C
C     history:        01/10/88--pdl by RGD
C
C     includes:
C
      EXTERNAL        DISKMGR_ERRCVTVAL              ! error converting value
      EXTERNAL        DISKMGR_ERRPUTCHR              ! error writing characters
      INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC' ! I/O definitions
C
C     arguments:
C
      INTEGER*4       VALUE                          ! value to be displayed
      INTEGER*4       ROW                            ! physical row of display
      INTEGER*4       COL                            ! physical column
      INTEGER*4       LENG                           ! number of chars to output
C
C     globals:
C

C
C     locals:
C
      INTEGER*4       STATUS                         ! function return status.
      CHARACTER*80    STRING                         ! output string
      INTEGER*4       ILENG
                      ! input leng argument.  (absolute value)
      INTEGER*4       JLENG                          ! actual input leng arg
C
C     functions:
C
      INTEGER*4       INSERT_INTEGER                 ! conversion routine
      INTEGER*4       SMG$PUT_CHARS                  ! SMG routines
C
C.....................................
C
C     Convert the integer value to a character.
C
      JLENG = LENG
      ILENG = LENG
      IF ( ILENG .LT. 0 ) ILENG = ILENG * -1
      STATUS = INSERT_INTEGER ( VALUE, STRING(1:ILENG), JLENG )
      IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'OUTPUT_INTEGER', STATUS,
     1                          %LOC(DISKMGR_ERRCVTVAL) )
        STRING(1:ILENG) = '*'
      END IF
```

```
C
C       Output the character string at the specified location.
C
        STATUS = SMG$PUT_CHARS ( DDID(2), STRING(1:ILENG), ROW, COL )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'OUTPUT_INTEGER', STATUS,
     1                             %LOC(DISKMGR__ERRPUTCHR) )
        END IF

        OUTPUT_INTEGER = STATUS

        RETURN
        END
```

A.29 OUTPUT_REAL

```
        INTEGER*4       FUNCTION OUTPUT_REAL ( VALUE, ROW, COL,
        1                                      LENG, RESOLUTION )
C
C       name:           Output real.
C
C       abstract:       Output a real value to the screen.
C
C       history:        01/11/87--pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR__ERRCVTVAL              ! error converting value
        EXTERNAL        DISKMGR__ERRPUTCHR              ! error writing display
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
C
C       arguments:
C
        REAL*4          VALUE                           ! value to be output
        INTEGER*4       ROW                             ! physical row of display
        INTEGER*4       COL                             ! physical column
        INTEGER*4       LENG                            ! number of chars to output
        INTEGER*4       RESOLUTION
                        ! number of chars to right of decimal point
C
C       globals:
C


C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        CHARACTER*80    STRING                          ! output string
C
C       functions:
```

```
C
      INTEGER*4       INSERT_REAL                    ! conversion routine
      INTEGER*4       SMG$PUT_CHARS                  ! SMG routine
C
C.....................................
C
C     Convert the real value to a character string.
C
      STATUS = INSERT_REAL ( VALUE, STRING(1:LENG), LENG, RESOLUTION )
      IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'OUTPUT_REAL', STATUS,
     1                          %LOC(DISKMGR__ERRCVTVAL) )
        STRING(1:LENG) = '*'
      END IF
C
C     Output the character string at the specified location
C
      STATUS = SMG$PUT_CHARS ( DDID(2), STRING(1:LENG), ROW, COL )
      IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'OUTPUT_REAL', STATUS,
     1                          %LOC(DISKMGR__ERRPUTCHR))
      END IF

      OUTPUT_REAL = STATUS

      RETURN
      END
```

A.30 OUTPUT_TOTAL_LINE

```
      INTEGER*4       FUNCTION OUTPUT_TOTAL_LINE ( OUTSTRING )
C
C     name:           Output total line
C
C     abstract:       Display the total line to the form.
C
C     history:        01/08/88--pdl by RGD
C
C     includes:
C
      INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
C
C     arguments:
C
      CHARACTER*80    OUTSTRING                       ! formatted output string
C
C     globals:
C

C
C     locals:
C
```

```
        INTEGER*4       STATUS                  ! function return status
        INTEGER*4       TOTALROW
        DATA            TOTALROW / 1 /          ! relative offset
        INTEGER*4       TOTALCOL
        DATA            TOTALCOL / 30 /         ! absolute offset
C
C       functions:
C
        INTEGER*4       SMG$SET_CURSOR_REL      ! SMG routines
        INTEGER*4       SMG$SET_CURSOR_ABS
        INTEGER*4       SMG$PUT_LINE
C
C.....................................
C
C
C       Output totals to the form.
C
        STATUS = SMG$SET_CURSOR_REL ( DDID(1), TOTALROW, )
        STATUS = SMG$SET_CURSOR_ABS ( DDID(1), , TOTALCOL )
        STATUS = SMG$PUT_LINE ( DDID(1), OUTSTRING(1:51) )

        OUTPUT_TOTAL_LINE = STATUS

        RETURN
        END
```

A.31 PARSE

```
        INTEGER*4       FUNCTION PARSE_COMMAND_LINE ( DEVICE, FORM,
        1                       NUM_DEVICES, UPDATE_RATE)
C
C       name:           Parse command line
C
C       abstract:       Parse the command line looking for the
C                       devices that are to be monitored.  Also,
C                       translates all logical name strings into
C                       physical device names and verifies that
C                       the specified device is indeed a disk.
C                       Valid qualifiers are:
C                         /DEVICE=(dev1, dev2, ..., devn) ==> list of devices to analyze.
C                         /FULL ==> use dev1 and start with full display instead of brief.
C                         /ALL  ==> automatically find all available disks.
C                         /EXCLUDE ==> ALL disks - those in the exclude category.
C                         /UPDATE=n ==> n = number of seconds to delay
C                                         before updating screen.
C
C       history:        11/01/87---pdl by RGD
C
C       includes:
C
        INCLUDE         'MS_INC:DISK_ANAL_CONST.INC'    ! constant definitions
```

```
C
C       arguments:
C
        CHARACTER*12    DEVICE( MAX_DEVICES )           ! list of devices to view
        INTEGER*2       FORM                            ! screen id to start
        INTEGER*2       NUM_DEVICES                     ! number of devices entered
        INTEGER*4       UPDATE_RATE                     ! delay between screen updates
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        CHARACTER*4     CUPDATE_RATE                    ! character string of update rate
        INTEGER*2       CLEN
                        ! length of character (device) string
C
C       functions:
C
        EXTERNAL        CLI$PRESENT, CLI$GET_VALUE, CLI$_ABSENT ! CLI definitions
        INTEGER*4       CLI$PRESENT, CLI$GET_VALUE, CLI$_ABSENT
        INTEGER*4       GET_ALL_DEVICES                 ! /ALL function
        INTEGER*4       GET_SPECIFIED_DEVICES           ! /DEVICE function
        INTEGER*4       REMOVE_EXCLUDED_DEVICES         ! /EXCLUDE function
C
C.....................................
C
C
C       Check if the /ALL qualifier was specified or if the
C       /DEVICES qualifier was specified.  One of the two must
C       be present or we will default to /ALL.
C
        IF ( CLI$PRESENT( 'ALL' ) ) THEN
           STATUS = GET_ALL_DEVICES ( DEVICE, MAX_DEVICES,
     1                                NUM_DEVICES )
        ELSE IF ( CLI$PRESENT ( 'DEVICES' ) ) THEN
           STATUS = GET_SPECIFIED_DEVICES ( DEVICE, MAX_DEVICES,
     1                                      NUM_DEVICES )
        ELSE ! default to /ALL
           STATUS = GET_ALL_DEVICES ( DEVICE, MAX_DEVICES,
     1                                NUM_DEVICES )
        END IF
C
C       Check if the /EXCLUDE qualifier was specified.
C
        IF ( CLI$PRESENT( 'EXCLUDE' ) ) THEN
           STATUS = REMOVE_EXCLUDED_DEVICES ( DEVICE, MAX_DEVICES,
     1                                        NUM_DEVICES )
        END IF
C
C       Check for /FULL qualifier.  If present, then start with form=2,
C       that is, the full device form.
```

```
C
        IF ( CLI$PRESENT( 'FULL' ) ) THEN
           FORM = 2
        ELSE
           FORM = 1
        END IF
C
C       Check for /UPDATE=n qualifier.  If the qualifier is not present,
C       or if a value is not specified, or if the specified value is
C       not valid, default to 5 seconds.
C
        IF ( CLI$PRESENT( 'UPDATE' ) ) THEN
           IF ( CLI$GET_VALUE('UPDATE', CUPDATE_RATE, CLEN )
     1          .EQ. %LOC( CLI$_ABSENT ) ) THEN
              CUPDATE_RATE = '5'
              CLEN = 1
           END IF
           DECODE(CLEN,100,CUPDATE_RATE(1:CLEN),ERR=110) UPDATE_RATE
100        FORMAT( I )
110        IF (UPDATE_RATE .LT. 1) UPDATE_RATE = 5
        ELSE
           UPDATE_RATE = 5
        END IF

        PARSE_COMMAND_LINE = 1!!SS$_NORMAL

        RETURN
        END
```

A.32 PROCESS_FKEY

```
        INTEGER*4       FUNCTION PROCESS_FKEY ( FORM, F_KEY )
C
C       name:           Process Function Key
C
C       abstract:       Determine which function key was entered.
C                       Each form has its own associated set of
C                       function keys.  Also, some function keys
C                       are available twice on the keyboard.  For
C                       example, the right arrow is defined as the
C                       right arrow and also as key pad key 6.
C                       If a function key is undefined, it is
C                       ignored.
C
C       return values:  -1 ==> change form
C                        0 ==> undefined function key
C                        1 ==> update token
C                        2 ==> refresh screen
C                        3 ==> help
C                        4 ==> do command
C
C       history:        01/03/88---pdl by RGD
```

```
C
C       includes:
C
        INCLUDE         '($SMGDEF)'                     ! key definitions
C
C       arguments:
C
        INTEGER*2       FORM                            ! current form being displayed
        INTEGER*2       F_KEY                           ! function key entered by user
C
C       globals:
C

C
C       locals:
C

C
C       functions:
C

C
C.......................................
C
C       Check for change form function key.
C
        IF (( F_KEY .EQ. SMG$K_TRM_CR ) .OR.
        1   ( F_KEY .EQ. SMG$K_TRM_ENTER ) .OR.
        2   ( F_KEY .EQ. SMG$K_TRM_NEXT_SCREEN ) .OR.
        3   ( F_KEY .EQ. SMG$K_TRM_PREV_SCREEN )) THEN
            PROCESS_FKEY = -1
            RETURN
        END IF
C
C       Check for refresh screen function key.
C
        IF (( F_KEY .EQ. SMG$K_TRM_KP7 ) .OR.
        1   ( F_KEY .EQ. SMG$K_TRM_CTRLW ) .OR.
        2   ( F_KEY .EQ. SMG$K_TRM_CTRLR )) THEN
            PROCESS_FKEY = 2
            RETURN
        END IF
C
C       Check for help function key.
C
        IF (( F_KEY .EQ. SMG$K_TRM_PF2 ) .OR.
        1   ( F_KEY .EQ. SMG$K_TRM_UPPERCASE_H )) THEN
            F_KEY = SMG$K_TRM_PF2
            PROCESS_FKEY = 3
            RETURN
        ELSE IF (( F_KEY .EQ. SMG$K_TRM_PF3 ) .OR.
        1        ( F_KEY .EQ. SMG$K_TRM_LOWERCASE_H )) THEN
            F_KEY = SMG$K_TRM_PF3
            PROCESS_FKEY = 3
```

```
            RETURN
          END IF
C
C         Check for do command function keys.
C
          IF (( F_KEY .EQ. SMG$K_TRM_PF4 ) .OR.
          1   ( F_KEY .EQ. SMG$K_TRM_DO )  .OR.
          2   ( F_KEY .EQ. SMG$K_TRM_UPPERCASE_R ) .OR.
          3   ( F_KEY .EQ. SMG$K_TRM_LOWERCASE_R )) THEN
              F_KEY = SMG$K_TRM_PF4
              PROCESS_FKEY = 4
              RETURN
          END IF
C
C         Define which keys have an equivalence meaning on all the forms.
C
          PROCESS_FKEY = 1
          IF (( F_KEY .EQ. SMG$K_TRM_KP6 )   .OR.
          1   ( F_KEY .EQ. SMG$K_TRM_RIGHT ) .OR.
          2   ( F_KEY .EQ. SMG$K_TRM_KP2 )   .OR.
          3   ( F_KEY .EQ. SMG$K_TRM_CTRLI ) .OR.
          3   ( F_KEY .EQ. SMG$K_TRM_DOWN )) THEN                ! next field
              F_KEY = SMG$K_TRM_DOWN
              RETURN
          ELSE IF (( F_KEY .EQ. SMG$K_TRM_KP4 )  .OR.
          1        ( F_KEY .EQ. SMG$K_TRM_LEFT ) .OR.
          2        ( F_KEY .EQ. SMG$K_TRM_BS )   .OR.
          3        ( F_KEY .EQ. SMG$K_TRM_CTRLH) .OR.
          4        ( F_KEY .EQ. SMG$K_TRM_KP8 )  .OR.
          5        ( F_KEY .EQ. SMG$K_TRM_UP )) THEN             ! prev field
              F_KEY = SMG$K_TRM_UP
              RETURN
          END IF

          PROCESS_FKEY = 0

          RETURN
          END
```

A.33 READ_BITMAP

```
          .TITLE  READ_BITMAP
;
;         This program will read the storage bitmap file into an extension of
;         process virtual address space.  it will then look at the string as
;         one long bit string. The GET_FREE_BLOCKS function will return
;         either a -1, or the number of continguos bits which represent
;         available disk space. This will serve as an indicator of
;         fragmentation of a disk volume.
;
;         SYMBOLIC DEFINITIONS
```

```
;
        $FABDEF
        $RABDEF

        .PSECT  DATA    PIC,NOEXE,LONG
        .ALIGN  LONG

INFAB:  $FAB                            ;FAB FOR BITMAP FILE
INRAB:  $RAB    FAB=INFAB,UBF=BUFFER,-
                USZ=512                 ;RAB FOR BITMAP FILE
BUFFER: .BLKB   512                     ;BUFFER FOR READ OF 1ST BITMAP BLOCK
FILNAM: .BLKB   80      ;STORAGE FOR BITMAP FILE SPEC
LENGTH: .BLKL   1       ;COPY OF DEVICE NAME STRING LENGTH
NAME:   .ASCII  /[0,0]BITMAP.SYS/       ;FILE SPEC FOR BITMAP
SIZE=.-NAME                             ;SIZE OF NAME STRING
CLUSTER:.BLKL   1                       ;FOR VOLUME CLUSTER SIZE
MAXBLOK:.BLKQ   1                       ;FOR MAX LOG BLOCK NUMBER
REM:    .BLKL   1                       ;COUNTERS
BITS:   .LONG   4096                    ;BITS IN ONE BLOCK
BYTES4: .BLKL   1
BLOCKS: .BLKL   1                       ;BLOCKS IN BITMAP FILE
VIRTUAL:.BLKL   1                       ;CONTAINS EXTENSION OF VAS
BIT_POS:.BLKL   1
FREE1:  .BLKL   1                       ;FIRST FREE BLOCK NUMBER
FREE2:  .BLKL   1                       ;END OF FREE BLOCK RANGE
RANGE:  .BLKL   1                       ;NUMBER OF FREE BLOCKS IN THIS RANGE

        .PSECT  CODE    PIC,NOWRT,LONG
        .ENTRY  OPEN_BITMAP, ^M<R6,R8,R11> ;ENTRY MASK
        MOVZWL  #0, LENGTH              ;Initialize local variables.
        MOVZWL  #0, CLUSTER
        MOVZWL  #0, MAXBLOK
        MOVZWL  #0, REM
        MOVZWL  #0, BYTES4
        MOVZWL  #0, BLOCKS
        MOVZWL  #0, VIRTUAL
        MOVZWL  #0, BIT_POS
        MOVZWL  #0, FREE1
        MOVZWL  #0, FREE2
        MOVZWL  #0, RANGE
        MOVZWL  #0, VIRTUAL
        MOVAL   @4(AP), R6              ;Pickup Adr of file name string arg.
        MOVZWL  (R6), LENGTH            ;SET DEVICE NAME SIZE IN LONGWORD
        addl2   #4, r6
        movl    (r6), r6
        MOVC3   LENGTH,(R6),FILNAM      ;SET DEVICE NAME IN BUFFER
        MOVAB   FILNAM,R6               ;GET ADDRESS OF BUFFER
        ADDL2   LENGTH,R6               ;ADD LENGTH JUST PUT IN THE BUFFER
        MOVC3   #SIZE,NAME,(R6)         ;SET FILE SPEC IN BUFFER
        MOVAB   FILNAM,INFAB+FAB$L_FNA  ;SET FILE NAME STRING
        ADDB3   LENGTH, #SIZE,-
                INFAB+FAB$B_FNS         ;SET FILE NAME STRING SIZE

        $OPEN   FAB=INFAB               ;OPEN THE BITMAP FILE
```

```
        BLBS    R0,10$                  ;CHECK STATUS
        BRW     END_OPEN
10$:    $CONNECT        RAB=INRAB       ;CONNECT RECORD STREAM
        BLBS    R0,20$                  ;CHECK STATUS
        BRW     END_OPEN
20$:    $GET    RAB=INRAB               ;READ IN THE FIRST BLOCK
        MOVZWL  BUFFER+2,CLUSTER        ;GET CLUSTER FACTOR
        MOVL    BUFFER+4, MAXBLCK       ;GET VOLUME SIZE

;       CALCULATIONS THAT MUST BE MADE BEFORE READING THE ACTUAL BITMAP
;
;       CLUSTERS-ON-DISK=  MAXBLK / CLUSTER SIZE (NUMBER OF BITS IN BITMAP)
;       BLOCKS-IN-BITMAP=  MAXBLK / 4096  (BITS IN ONE BLOCK)
;       BYTES NEEDED    =  BLOCKS * 512   (BYTES IN ONE BLOCK)
;
        DIVL2   CLUSTER,MAXBLCK ;NUMBER OF CLUSTERS (BITS IN BITMAP)
        EDIV    BITS,MAXBLCK,BLOCKS,REM;FIGURE OUT NUMBER OF BLOCKS IN FILE
        TSTL    REM                     ;WAS THERE A REMAINDER FROM THE DIVIDE?
        BEQL    NO_REM                  ;BRANCH IF NO REMAINDER
        INCL    BLOCKS                  ;IF SO, ROUND BLOCK COUNT UP
NO_REM: MULL3   #512,BLOCKS,BYTES4      ;NUMBER OF BYTES NEEDED

;       NOW ALLOCATED VIRTUAL MEMORY TO WRITE THE CONTENTS OF THE REST
;       OF THE BITMAP, SO WE WON'T BE TYING UP THE FILE
;
        PUSHAL  VIRTUAL                 ;GET ADDITIONAL BYTES FOR BITMAP BLOCKS
        PUSHAL  BYTES4                  ;BYTES NEEDED
        CALLS   #2,G^LIB$GET_VM
        BLBS    R0,10$                  ;CHECK STATUS
        BRW     END_OPEN
10$:    MOVL    VIRTUAL,INRAB+RAB$L_UBF ;SET ADDRESS OF NEW USER BUFFER
        MOVL    BLOCKS,R11              ;SET NUMBER OF BLOCKS TO COPY
LOOP:   $GET    RAB=INRAB               ;COPY BITMAP BLOCK TO VIRT MEMORY
        MOVL    R11,RANGE
        ADDL2   #512,INRAB+RAB$L_UBF    ;SET NEXT BLOCK POINTER
        DECL    R11
        tstl    R11                     ;COPIED ALL BLOCKS ?
        BGTR    LOOP                    ;COPIED ALL BLOCKS ?
        CLRL    BIT_POS
        $CLOSE  FAB=INFAB               ;YES, CLOSE BITMAP FILE
end_open:
        movl    r0, @8(ap)
        movzwl  #0, bit_pos             ;ZERO COUNT EACH TIME.
        RET
;
        .entry  get_free_blocks, ^M<R6,R8,R11>
        MOVL    VIRTUAL,R11             ;SET ADDRESS OF VIRT MEMORY
        MULL3   #4096,BLOCKS,BITS       ;TOTAL BITS TO READ
        MOVL    BIT_POS, R8             ;USE R8 AS POINTER REGISTER
FFS_LOOP:
        FFS     R8,#32,(R11),R8         ;LOOK FOR 1ST SET BIT, PUT INTO R8
        BNEQ    FFS_LOOP_EXIT           ;BRANCH WHEN WE FIND ONE SET
        CMPL    MAXBLCK,R8              ;HAVE WE READ ALL BITS YET ?
        BLEQ    END_GET                 ;YES, EXIT OUT OF HERE
```

```
        BRB     FFS_LOOP
FFS_LOOP_EXIT:
        MULL3   CLUSTER,R8,FREE1        ;GET LOGICAL BLOCK NUMBER OF FREE BLOCK
FFC_LOOP:
        FFC     R8,#32,(R11),R8         ;NO, FIND CLEAR BIT (ALLOCATED BLOCK)
        BNEQ    FFC_LOOP_EXIT           ;BRANCH WHEN WE FIND ONE
        CMPL    MAXBLCK,R8              ;HAVE WE READ ALL BITS YET ?
        BLEQ    END_GET                 ;YES, EXIT OUT OF HERE
        BRB     FFC_LOOP
FFC_LOOP_EXIT:
        MULL3   CLUSTER,R8,FREE2        ;GET LOGICAL BLOCK NUMBER OF ALLOC BLK
        SUBL3   FREE1,FREE2,RANGE       ;GET THE NUMBER OF BLOCKS IN THIS RANGE
        BRB     DONE                    ;YES, EXIT OUT OF HERE
END_GET:MNEGL   #1, RANGE
DONE:   MOVL    RANGE, R0
        MOVL    R8, BIT_POS
        RET
;
        .ENTRY  CLOSE_BITMAP, ^M<R6,R8,R11>
        PUSHAL  VIRTUAL                 ;FREE BYTES FOR BITMAP
        PUSHAL  BYTES4
        CALLS   #2,G^LIB$FREE_VM
        RET
        .END
```

A.34 READ_FKEY

```
        INTEGER*4       FUNCTION READ_FKEY ( F_KEY, WAIT_TIME, FORM )
C
C       name:           Read Function Key
C
C       abstract:       Reads the next keystroke entered by the
C                       user at the keyboard if the key is pressed
C                       before WAIT_TIME seconds expires. If
C                       timeout occurs, the routine returns
C                       SS$_TIMEOUT as its return status.
C
C       history:        12/01/87---pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR_ERREADKSK       ! error reading keystroke
        INCLUDE         '($SMGDEF)'             ! Screen Management definitions
        INCLUDE         '($SSDEF)'              ! return status definitions
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! IO definitions
C
C       arguments:
C
        INTEGER*2       F_KEY                   ! key entered by user
        INTEGER*4       WAIT_TIME               ! time out period in seconds
        INTEGER*2       FORM                    ! current screen id
C
```

```
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! return status
C
C       functions:
C
        INTEGER*4       SMG$READ_KEYSTROKE              ! SMG routine
C
C.....................................
C
C       Read with timeout, the next key entered by the user.
C
        STATUS = SMG$READ_KEYSTROKE ( VKID,             ! virtual keyboard id
        1                             F_KEY,            ! key entered
        2                             ,WAIT_TIME,       ! delay period
        3                             DDID(FORM) ,, )   ! virtual display id

        IF (( .NOT. STATUS ) .AND. ( STATUS .NE. SS$_TIMEOUT )) THEN
           CALL REPORT_EXCEPTION ( 'READ_FKEY', STATUS,
        1                          %LOC(DISKMGR__ERREADKSK) )
        END IF
        READ_FKEY = STATUS
        RETURN
        END


A.35  REDRAW_FORM1

        INTEGER*4       FUNCTION REDRAW_FORM1 ( NUM_DEVICES )
C
C       name:           Redraw form 1
C
C       abstract:       Redraw the body of form 1 that is considered
C                       the background of the form.
C
C       history:        11/01/87--pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR__ERRPUTCHR
                        ! error writing characters to display
        INCLUDE         'MS_INC:FORM_BCKGRND_DEFN.INC'  ! form background definitions
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O parameters for SMG calls
C
C       arguments:
C
        INTEGER*2       NUM_DEVICES
                        ! number of devices to be displayed
C
```

```
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        INTEGER*4       I                               ! temporary counter
C
C       functions:
C
        INTEGER*4       SMG$PUT_LINE                    ! SMG routines
        INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY
C
C.....................................
C
C
C       Redraw the body of form 1, i.e., form 1's body
C       consists of a subheader for the field labels,
C       blank lines for the device output, and a total
C       section at the bottom of the form.
C
        STATUS = SMG$PUT_LINE ( DDID(1), SUBHEADER(1) )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                           %LOC(DISKMGR__ERRPUTCHR) )
        END IF

        STATUS = SMG$PUT_LINE ( DDID(1), SUBHEADER(2) )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                           %LOC(DISKMGR__ERRPUTCHR) )
        END IF

        STATUS = SMG$PUT_LINE ( DDID(1), SUBHEADER(3) )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                           %LOC(DISKMGR__ERRPUTCHR) )
        END IF

        DO I = 1, NUM_DEVICES
           STATUS = SMG$PUT_LINE ( DDID(1), BLANKLINE )
           IF ( .NOT. STATUS ) THEN
              CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                              %LOC(DISKMGR__ERRPUTCHR) )
           END IF
        END DO

        STATUS = SMG$PUT_LINE ( DDID(1), TOTAL(1) )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                           %LOC(DISKMGR__ERRPUTCHR) )
        END IF
```

```
      STATUS = SMG$PUT_LINE ( DDID(1), TOTAL(2) )
      IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                          %LOC(DISKMGR__ERRPUTCHR) )
      END IF

      !! May need to fill rest of display with blank lines
C
C     Finally, make the form visible to the user.
C
      STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(1), PBID, 1, 1 )
      IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'REDRAW_FORM1', STATUS,
     1                          %LOC(DISKMGR__ERRPUTCHR) )
      END IF

      REDRAW_FORM1 = STATUS

      RETURN
      END
```

A.36 REDRAW_FORM2

```
      INTEGER*4       FUNCTION REDRAW_FORM2 ( )
C
C     name:           Redraw form 2
C
C     abstract:       Redraw the body of form 2
C
C     history:        11/01/87--pdl by RGD
C
C     includes:
C
      EXTERNAL        DISKMGR__ERRPUTCHR
                      ! error writing characters to display
      EXTERNAL        DISKMGR__ERRRESSCR          ! error restoring screen display
      INCLUDE         'MS_INC:FORM_BCKGRND_DEFN.INC'  ! form background definitions
      INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O parameters for SMG calls
C
C     arguments:
C

C
C     globals:
C

C
C     locals:
C
      INTEGER*4       STATUS                      ! function return value
      INTEGER*4       I                           ! temporary counter
C
```

```
C       functions:
C
        INTEGER*4       SMG$PUT_LINE                    ! SMG routines
        INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY
C
C . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
C
C
C       Output all remaining lines of background text to the display.
C
        DO I = 1, 20
           STATUS = SMG$PUT_LINE ( DDID(2), LINE(I) )
           IF ( .NOT. STATUS ) THEN
              CALL REPORT_EXCEPTION ( 'REDRAW_FORM2', STATUS,
     1                                %LOC(DISKMGR__ERRPUTCHR) )
           END IF
        END DO

        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(2), PBID, 1, 1 )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_FORM2', STATUS,
     1                                %LOC(DISKMGR__ERRRESSCR) )
        END IF

        REDRAW_FORM2 = STATUS

        RETURN
        END
```

A.37 REDRAW_HEADER

```
        INTEGER*4       FUNCTION REDRAW_HEADER ( FORM )
C
C       name:           Redraw header
C
C       abstract:       Redraws the header of the current form.
C
C       history:        11/01/87--pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR__ERRREMDIS              ! error removing display
        EXTERNAL        DISKMGR__ERRPUTCHR              ! error writing chars
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
        INCLUDE         'MS_INC:HEADER_BCKGRND_DEFN.INC'! header definitions
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O parameters for SMG calls
C
C       arguments:
C
        INTEGER*2       FORM                            ! current screen id
C
C       globals:
```

```
C
C
C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
C
C       functions:
C
        INTEGER*4       SMG$UNPASTE_VIRTUAL_DISPLAY     ! SMG routines
        INTEGER*4       SMG$SET_CURSOR_ABS
        INTEGER*4       SMG$PUT_LINE
C
C.......................................
C
C
C       Remove current version of either virtual display if it exists.
C
        STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(1), PBID )
        IF (( .NOT. STATUS ) .AND. ( STATUS .NE. SMG$_INVARG )) THEN
C
C          Above check should exclude an error message of
C          "The specified virtual display is not pasted to
C           the specified pasteboard."  However, it does not work??
C          The error code returned in VMS V4.5, 4.6, and 4.6
C          is 1213044 decimal.
C
           IF ( STATUS .NE. 1213044 ) THEN
              CALL REPORT_EXCEPTION ( 'REDRAW_HEADER', STATUS,
     1                                %LOC(DISKMGR__ERRREMDIS) )
           END IF
        END IF

        STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(2), PBID )
        IF (( .NOT. STATUS ) .AND. ( STATUS .NE. SMG$_INVARG )) THEN
           IF ( STATUS .NE. 1213044 ) THEN      ! same as above
              CALL REPORT_EXCEPTION ( 'REDRAW_HEADER', STATUS,
     1                                %LOC(DISKMGR__ERRREMDIS) )
           END IF
        END IF
C
C       Fill header display with its required data, i.e.,
C       the header should consist of a title line, the date and time
C       line, and a blank third line of the form.
C
        STATUS = SMG$SET_CURSOR_ABS ( DDID(FORM), 1, 1 )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_HEADER', STATUS,
     1                             %LOC(DISKMGR__ERRPUTCHR) )
        END IF

        STATUS = SMG$PUT_LINE ( DDID(FORM), TITLE )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'REDRAW_HEADER', STATUS,
```

```
     1                          %LOC(DISKMGR_ERRPUTCHR) )
     END IF

     STATUS = SMG$PUT_LINE ( DDID(FORM), DATETIME )
     IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'REDRAW_HEADER', STATUS,
     1                          %LOC(DISKMGR_ERRPUTCHR) )
     END IF

     STATUS = SMG$PUT_LINE ( DDID(FORM), BLANKLINE )
     IF ( .NOT. STATUS ) THEN
        CALL REPORT_EXCEPTION ( 'REDRAW_HEADER', STATUS,
     1                          %LOC(DISKMGR_ERRPUTCHR) )
     END IF

     REDRAW_HEADER = STATUS

     RETURN
     END
```

A.38 REDRAW_SCREEN

```
     INTEGER*4      FUNCTION REDRAW_SCREEN ( FORM, NUM_DEVICES )
C
C    name:          Redraw Screen
C
C    abstract:      Redraw the static portion of the screen that
C                   the operator is currently viewing.  This
C                   consists of the text background in the header
C                   and the body of the form.
C
C    history:       11/01/87--pdl by RGD
C
C    includes:
C

C
C    arguments:
C
     INTEGER*2      FORM                          ! current screen id
     INTEGER*2      NUM_DEVICES
                    ! number of devices to be displayed
C
C    globals:
C

C
C    locals:
C
     INTEGER*4      STATUS
C
C    functions:
```

```
C
      INTEGER*4       REDRAW_HEADER                 ! redraws header
      INTEGER*4       REDRAW_FORM1                  ! redraws form 1
      INTEGER*4       REDRAW_FORM2                  ! redraws form 2
C
C.....................................
C
C
C     Redraw header of the display.
C
      STATUS = REDRAW_HEADER ( FORM )
C
C     Based on which is the current form, redraw its body.
C
      IF ( FORM .EQ. 1 ) THEN
         STATUS = REDRAW_FORM1 ( NUM_DEVICES )
      ELSE ! form = 2
         STATUS = REDRAW_FORM2 ( )
      END IF

      REDRAW_SCREEN = STATUS

      RETURN
      END
```

A.39 REMOVE_EXCLUDED_DEVICES

```
C
C.....................................
C
      INTEGER*4       FUNCTION REMOVE_EXCLUDED_DEVICES (
      1                                             DEVICE,
      2                                             MAX_DEVICES,
      3                                             NUM_DEVICES )
C
C     name:           Remove Excluded Devices
C
C     abstract:       Using the /EXCLUDE qualifier, implies that
C                     the user wishes NOT to monitor the excluded
C                     devices.  This routine searches the current
C                     list of devices, removing those that are
C                     in the exclude list.
C
C     history:        02/20/88---pdl by RGD
C
C     includes:
C
      INCLUDE         '($SSDEF)'                    ! system service definitions
C
C     arguments:
C
```

```
      INTEGER*2       MAX_DEVICES
                      ! max # devices that can be monitored
      CHARACTER*12    DEVICE ( MAX_DEVICES )          ! list of devices to monitor
      INTEGER*2       NUM_DEVICES                     ! number of devices entered
C
C     globals:
C


C
C     locals:
C
      CHARACTER*12    DEV
                      ! device name off the exclude list.
      INTEGER*2       CLEN                            ! length of character string
      INTEGER*4       STATUS                          ! function return status
      INTEGER*2       I                               ! temporary array index
      INTEGER*2       J                               ! ditto
C
C     functions:
C
      EXTERNAL        CLI$GET_VALUE, CLI$_ABSENT      ! CLI definitions
      INTEGER*4       CLI$GET_VALUE, CLI$_ABSENT
      INTEGER*4       TRANSLATE_DEVNAM                ! logical name translation routine
C
C.....................................
C
C
C     Read each of the devices in the list.
C     Qualifier has the form:  /EXCLUDE=(device1, device2, ... )
C
      DO WHILE ( CLI$GET_VALUE( 'EXCLUDE', DEV,
     1                          CLEN ) .NE. %LOC( CLI$_ABSENT ) )
C
C        Translate the device name until we get the
C        physical device name.
C
         STATUS = TRANSLATE_DEVNAM ( DEV )
         IF ( STATUS .NE. SS$_NOTRAN ) THEN
            REMOVE_EXCLUDED_DEVICES = STATUS
            RETURN
         END IF
C
C        Eliminate this device if found in the current device list.
C
         DO I = 1, NUM_DEVICES
            IF ( DEV .EQ. DEVICE(I) ) THEN
               GOTO 10
            END IF
         END DO
C
C        If we fall out of the above loop, we did not match the
C        device name.
C
         GOTO 100
```

```
C
C          At label 10, device name was found, therefore, remove
C          it from our current list.
10         CONTINUE
           DO J = (I+1), NUM_DEVICES
              DEVICE (J-1) = DEVICE(J)
           END DO
           NUM_DEVICES = NUM_DEVICES - 1
100        CONTINUE
        END DO

        REMOVE_EXCLUDED_DEVICES = 1

        RETURN
        END
```

A.40 REPORT_EXCEPTION

```
        SUBROUTINE      REPORT_EXCEPTION ( MODULE_NAME,
        1                                  ESTATUS, DSTATUS )
C
C       name:           Report Exception
C
C       abstract:       Displays an error message on the bottom of
C                       the terminal for viewing by the user.
C
C       history:        01/01/88--pdl by RGD
C
C       includes:
C
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! I/O definitions
C
C       arguments:
C
        CHARACTER*(*)   MODULE_NAME                     ! module reporting error
        INTEGER*4       ESTATUS                         ! system error status
        INTEGER*4       DSTATUS                         ! DISKMGR error status
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        CHARACTER*132   OUTSTRING                       ! formatted error message
        CHARACTER*256   MSG                             ! associated ESTATUS message
        INTEGER*4       MSGLEN                          ! length returned by GETMSG
        INTEGER*2       I                               ! index counter
C
C       functions:
```

```
C
      INTEGER*4       INSERT_INTEGER
                      ! inserts integer value into output string
      INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY       ! SMG routines
      INTEGER*4       SMG$UNPASTE_VIRTUAL_DISPLAY
      INTEGER*4       SMG$PUT_CHARS
      INTEGER*4       SMG$ERASE_DISPLAY
      INTEGER*4       SYS$GETMSG
C
C.....................................
C
C     Remove the current message line from the display.
C
      STATUS = SMG$UNPASTE_VIRTUAL_DISPLAY ( DDID(5), PBID )
C
C     Erase the old text in the display.
C
      STATUS = SMG$ERASE_DISPLAY ( DDID(5) )
C
C     Create the output string of the error message.
C
      DO I = 1, 256
         MSG(I:I) = CHAR ( 0 )                        ! fill with null chars
      END DO
      OUTSTRING = ' '                                 ! blank output string
      OUTSTRING(2:17) = MODULE_NAME
C
C     Get message associate with the DSTATUS code.
C
      STATUS = SYS$GETMSG ( %VAL(DSTATUS),
     1                      MSGLEN,
     2                      %DESCR(MSG),
     3                      %VAL(1), )
      OUTSTRING(18:56) = MSG
C
C     Get message associated with the ESTATUS code.
C     ESTATUS is the VMS message code.  Under V1.0 of DISKMGR,
C     this message will not be displayed.
C
!     STATUS = SYS$GETMSG ( %VAL(ESTATUS),
!    1                      MSGLEN,
!    2                      %DESCR(MSG),
!    3                      %VAL(1), )
!     OUTSTRING(57:132) = MSG(1:MSGLEN) // ' '
C
C     Output the error message to the display.
C
      STATUS = SMG$PUT_CHARS ( DDID(5), OUTSTRING(1:132), 1, 1, ,
     1                         SMG$M_REVERSE )
C
C     Paste the display to the pasteboard.
C
      STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(5), PBID, 24, 1 )
```

```
        RETURN
        END
```

A.41 SET_ITEMLIST

```
        INTEGER*4       FUNCTION SET_ITEMLIST ( ITEMLIST,
        1                                       DEV_BRIEF,
        2                                       DEV_FULL,
        3                                       FORM )
C
C       name:           SET ITEMLIST
C
C       abstract:       Sets the appropriate values of the itemlist
C                       depending on the type of form GET_DEVICE_INFO
C                       is using to get information.
C
C       history:        01/01/88---pdl by RGD
C
C       includes:
C
        INCLUDE         'MS_INC:ITMLST_STRUCT_DEFN.INC' ! item list structure definitions
        INCLUDE         'MS_INC:DEV_ATT.INC'
                        ! device attributes structure definition
        INCLUDE         '($DVIDEF)'                     ! GETDVI symbol definitions
C
C       arguments:
C
        RECORD  /ITMLST/        ITEMLIST ( MAX_ITEMS + 1 )
        RECORD  /DEV_BRIEF_ATT/ DEV_BRIEF
        RECORD  /DEV_FULL_ATT/  DEV_FULL
        INTEGER*2               FORM
C
C       globals:
C

C
C       locals:
C

C
C       functions:
C

C
C.......................................
C
        IF ( FORM .EQ. 1 ) THEN
           ITEMLIST ( 8 ).END_LIST = 0
        ELSE
           ITEMLIST ( 8 ).BUFFER_LENGTH = 10
           ITEMLIST ( 8 ).ITEM_CODE = DVI$_MEDIA_NAME
        END IF
```

```
        SET_ITEMLIST = 1
        RETURN
        END
```

A.42 START_IO

```
        INTEGER*4       FUNCTION START_IO ( FORM )
C
C       name:           Start Input/Output
C
C       abstract:       Pastes initial display to the screen
C
C       history:        11/01/87---pdl by RGD
C
C       includes:
C
        EXTERNAL        DISKMGR_ERRPUTCHR               ! error unpasting display
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! global IO definitions
        INCLUDE         'MS_INC:DISK_ANAL_CONST.INC'    ! constant definitions
        INCLUDE         '($SMGDEF)'                     ! SMG definitions
C
C       arguments:
C
        INTEGER*2       FORM                            ! current screen id
C
C       globals:
C
C
C       locals:
C
        INTEGER*4       STATUS                          ! return status
C
C       functions:
C
        INTEGER*4       SMG$PASTE_VIRTUAL_DISPLAY       ! SMG routines
C
C.....
C
C
C
C       Paste the initially blank virtual display to the pasteboard.
C
        STATUS = SMG$PASTE_VIRTUAL_DISPLAY ( DDID(FORM), PBID, 1, 1 )
        IF ( .NOT. STATUS ) THEN
           CALL REPORT_EXCEPTION ( 'START_IO', STATUS,
     1                             %LOC(DISKMGR_ERRPUTCHR) )
        END IF
        START_IO = STATUS
        RETURN
        END
```

A.43 TRANSLATE_DEVNAM

```
      INTEGER*4       FUNCTION TRANSLATE_DEVNAM ( DEVICE )
C
C     name:           Translate Device Name
C
C     abstract:       Search all logical name tables and return
C                     the translation of the given device name.
C
C     history:        02/01/88--pdl by RGD
C
C     includes:
C
      INCLUDE         '($SSDEF)'                  ! system service definitions
C
C     arguments:
C
      CHARACTER*(*)   DEVICE                      ! given device name
C
C     globals:
C

C
C     locals:
C
      BYTE            TABLE                       ! table where name is found
      BYTE            ACC_MODE                    ! access mode to find logical name
      BYTE            DSB_MSK                     ! disable search mode mask
      INTEGER*4       DST_LEN                     ! length of string returned
      CHARACTER*20    LOGICAL_NAME                ! logical name to search for
      CHARACTER*79    TRANSLATION                 ! translation of logical name
      INTEGER*4       STATUS                      ! function return status
      INTEGER*2       J                           ! temporary array index
      INTEGER*2       I
      INTEGER*2       MAX_TRANSLATIONS
      PARAMETER       ( MAX_TRANSLATIONS = 20 )
C
C     functions:
C
      INTEGER*4       LIB$SYS_TRNLOG
C
C.....................................
C
C

      LOGICAL_NAME = ' '
      LOGICAL_NAME = DEVICE
C
C     This example searches for LOGICAL_NAME and returns the logical
C     translation for it.  It also passes all parameters to the
C     run time library function.
C
C
C     Arguments LOGICAL_NAME and DSB_MSK are set by the user,
C     all other arguments are returned by the RTL call.
```

```
C
        STATUS = SS$_NORMAL
        I = 0
        DO WHILE ((( STATUS .NE. SS$_NOTRAN ) .AND. ( STATUS )) .AND.
        1         ( I .LT. MAX_TRANSLATIONS ))
           I = I + 1
           DO J = 20, 1, -1
              IF ( LOGICAL_NAME(J:J) .NE. ' ' ) GOTO 30
           END DO
           J = 1
30         CONTINUE

           DSB_MSK = 0          ! search all tables

           STATUS = LIB$SYS_TRNLOG ( LOGICAL_NAME(1:J), DST_LEN,
        1                            TRANSLATION, TABLE, ACC_MODE,
        2                            DSB_MSK )
           IF ( .NOT. STATUS ) THEN
              TRANSLATE_DEVNAM = STATUS
              RETURN
           END IF

           LOGICAL_NAME = ' '
           LOGICAL_NAME(1:DST_LEN) = TRANSLATION(1:DST_LEN)

        END DO

        DEVICE = ' '
        DEVICE = LOGICAL_NAME

        TRANSLATE_DEVNAM = STATUS

        RETURN
        END
```

A.44 UPDATE_DYNAMIC

```
        INTEGER*4       FUNCTION UPDATE_DYNAMIC ( FORM,
        1                                         NUM_DEVICES, DEV_BRIEF,
        2                                         DEV_FULL, TOKEN,
        3                                         DEV_NUM )
C
C       name:           Update Dynamic
C
C       abstract:       Update the dynamic portion of the screen
C                       the operator is currently viewing.  These
C                       values are the ones that can change on
C                       each update of the screen
C
C       history:        11/01/87---pdl by RGD
```

```
C
C       includes:
C
        INCLUDE         '($DVIDEF)'                     ! device characteristics
        INCLUDE         '($DEVDEF)'                     ! device characteristics
        INCLUDE         'MS_INC:DISK_ANAL_CONST.INC'    ! constant definitions
        INCLUDE         'MS_INC:DEV_ATT.INC'
                        ! device attributes structure definition
        INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'  ! IO definitions
        INCLUDE         'MS_INC:TOKEN_DEFN.INC'         ! token definitions
C
C       arguments:
C
        INTEGER*2       FORM                            !current screen id
        INTEGER*2       NUM_DEVICES
                        ! actual num devices being displayed
        RECORD          /DEV_BRIEF_ATT/ DEV_BRIEF ( MAX_DEVICES )
        RECORD          /DEV_FULL_ATT/  DEV_FULL
        RECORD          /TOKEN/ TOKEN                   ! current field marker
        INTEGER*2       DEV_NUM                         ! device index for form 2
C
C       globals:
C

C
C       locals:
C
        INTEGER*4       STATUS                          ! return status
        INTEGER*4       END                             ! end of specified substring
        CHARACTER*23    ASCII_TIME                      ! current time as ascii string
        INTEGER*4       TIMEROW                         ! row to display time
        DATA            TIMEROW / 2 /
        INTEGER*4       TIMECOL                         ! column to display time
        DATA            TIMECOL / 29 /
        INTEGER*4       I                               ! index variable
        CHARACTER*80    OUTSTRING                       ! output string
        INTEGER*4       FIRSTROW(2)                     ! firstline position of each form
        DATA            FIRSTROW / 7, 4 /
        INTEGER*4       FIRSTCOL(2)                     ! column of first line
        DATA            FIRSTCOL / 1, 1 /
        INTEGER*4       T_ERROR_COUNT                   ! totals
        INTEGER*4       T_MAX_BLOCKS
        INTEGER*4       T_FREE_BLOCKS
        INTEGER*4       T_TRANS_COUNT
        INTEGER*4       T_MOUNT_COUNT
        INTEGER*4       ITEMP                           ! temporary variable
        REAL*4          R_MAX_BLOCKS                    ! real values for real arithmetic
        REAL*4          R_FREE_BLOCKS
        REAL*4          PERCENT_USED
C
C       functions:
C
        INTEGER*4       CREATE_BRIEF_LINE               ! formats brief display line
        INTEGER*4       CREATE_TOTAL_LINE               ! formats total display line
```

```
      INTEGER*4       OUTPUT_TOTAL_LINE              ! outputs total dislay line
      INTEGER*4       GET_TIME                       ! obtain current time
      INTEGER*4       OUTPUT_CHARS                   ! output a character string
      INTEGER*4       OUTPUT_INTEGER
                      ! output an integer value to display
      INTEGER*4       OUTPUT_REAL
                      ! output a real value to the display
      INTEGER*4       GET_PROT                       ! returns protection code string
      INTEGER*4       SMG$BEGIN_DISPLAY_UPDATE       ! define SMG routines
      INTEGER*4       SMG$END_DISPLAY_UPDATE
      INTEGER*4       SMG$SET_CURSOR_ABS
      INTEGER*4       SMG$PUT_LINE
C
C.....................................
C
C     Control the screen updates such that output operations
C     to the pasted display are reflected only in the display's
C     buffers.  Then, once all modifications have been made,
C     these buffers are written to the screen.
C
C
C     Update the header of the form; i.e., update the time
C
      STATUS = SMG$BEGIN_DISPLAY_UPDATE ( DDID(FORM) )
      STATUS = GET_TIME ( ASCII_TIME )
      STATUS = SMG$SET_CURSOR_ABS ( DDID(FORM), TIMEROW, TIMECOL )
      STATUS = SMG$PUT_LINE ( DDID(FORM), ASCII_TIME )
      STATUS = SMG$SET_CURSOR_ABS ( DDID(FORM),
     1                              FIRSTROW(FORM), FIRSTCOL(FORM) )
C
C     Update dynamic portion of screen based on value of form.
C
      IF ( FORM .EQ. 1 ) THEN
C
C        For each device, output one brief status line.
C
         DO I = 1, NUM_DEVICES
            STATUS = CREATE_BRIEF_LINE ( DEV_BRIEF(I), OUTSTRING )
            IF (TOKEN.FIELD_.EQ. I) OUTSTRING(1:1) = TOKEN.CHR
            STATUS = SMG$PUT_LINE ( DDID(1), OUtSTRING(1:80) )
         END DO
C
C        Total all of the displayed devices and output one
C        total line for all the devices.
C
         CALL GET_TOTALS ( T_ERROR_COUNT, T_MAX_BLOCKS,
     1                     T_FREE_BLOCKS, T_TRANS_COUNT,
     2                     T_MOUNT_COUNT, DEV_BRIEF(1), NUM_DEVICES )
         STATUS = CREATE_TOTAL_LINE ( T_ERROR_COUNT, T_MAX_BLOCKS,
     1                                T_FREE_BLOCKS, T_TRANS_COUNT,
     2                                T_MOUNT_COUNT, OUTSTRING )
         STATUS = OUTPUT_TOTAL_LINE ( OUTSTRING )

      ELSE    ! form = 2
```

```
      I = DEV_NUM
C
C     Output device name, volume label, media type, files-11 type,
C     and device status.
C
      END = MIN(13,DEV_BRIEF(I).DEV_NAM_LEN)
      STATUS = OUTPUT_CHARS ( DEV_BRIEF(I).DEVICE_NAME(2:END),
     1                        END-1, 4, 2 )

      END = MIN(12,DEV_BRIEF(I).DEV_VOL_NAME_LEN)
      STATUS = OUTPUT_CHARS ( DEV_BRIEF(I).VOLUME_LABEL,
     1                        END, 4, 16 )

      STATUS = OUTPUT_CHARS ( DEV_FULL.MEDIA_NAME,
     1                        DEV_FULL.MEDNAM_LEN, 4, 30 )

      OUTSTRING = ' '
      IF ( DEV_FULL.ACP_TYPE .EQ. DVI$C_ACP_F11V1 ) THEN
         OUTSTRING(1:16) = 'Files-11 Level 1'
      ELSE IF (DEV_FULL.ACP_TYPE .EQ. DVI$C_ACP_F11V2 ) THEN
         OUTSTRING(1:16) = 'Files-11 Level 2'
      ELSE ! unknown for disk
         OUTSTRING(1:16) = 'Unknown format  '
      END IF
      STATUS = OUTPUT_CHARS ( OUTSTRING(1:16), 16, 4, 36)

      IF ( DEV_FULL.MOUNTED ) THEN
         OUTSTRING(1:10) = 'mounted'
      ELSE
         OUTSTRING(1:10) = 'dismounted'
      END IF
      STATUS = OUTPUT_CHARS ( OUTSTRING(1:10), 10, 4, 55 )
C
C     Output occurs on the rest of form by parts.
C     Output owner PID, its UIC, and the device protection code.
C
      STATUS = OUTPUT_INTEGER ( DEV_FULL.PID, 5, 13, -8 )
      ITEMP = DEV_FULL.OWNUIC.GROUP
      STATUS = OUTPUT_INTEGER ( ITEMP, 5, 29, -3 )
      ITEMP = DEV_FULL.OWNUIC.MEMBER
      STATUS = OUTPUT_INTEGER ( ITEMP, 5, 33, -3 )
      OUTSTRING = ' '
      STATUS = GET_PROT( DEV_FULL.PROT, OUTSTRING(1:28) )
      STATUS = OUTPUT_CHARS ( OUTSTRING(1:28), 28, 5, 49 )
C
C     Output max blocks, free blocks, blocks used, and percent used.
C
      STATUS = OUTPUT_INTEGER ( DEV_BRIEF(I).MAX_BLOCKS, 6, 12, 7 )
      STATUS = OUTPUT_INTEGER ( DEV_BRIEF(I).FREE_BLOCKS,7, 12, 7 )
      ITEMP = DEV_BRIEF(I).MAX_BLOCKS - DEV_BRIEF(I).FREE_BLOCKS
      STATUS = OUTPUT_INTEGER ( ITEMP,                   8, 12, 7 )
      R_MAX_BLOCKS  = DEV_BRIEF(I).MAX_BLOCKS
      R_FREE_BLOCKS = DEV_BRIEF(I).FREE_BLOCKS
      PERCENT_USED  = ((R_MAX_BLOCKS-R_FREE_BLOCKS)/R_MAX_BLOCKS)
```

```
     1              * 100.0
          STATUS = OUTPUT_REAL ( PERCENT_USED,  9, 14, 5, 1 )
C
C         Ouptut mount count, transaction count, reference count, and error count.
C
          STATUS = OUTPUT_INTEGER ( DEV_BRIEF(I).MOUNT_COUNT, 6, 33, 5)
          STATUS = OUTPUT_INTEGER ( DEV_BRIEF(I).TRANS_COUNT, 7, 33, 5)
          STATUS = OUTPUT_INTEGER ( DEV_FULL.REF_COUNT,       8, 33, 5)
          STATUS = OUTPUT_INTEGER ( DEV_BRIEF(I).ERROR_COUNT, 9, 33, 5)
C
C         Output number of cylinders, number of tracks/cylinder,
C         number sectors/track, and device cluster factor size.
C
          STATUS = OUTPUT_INTEGER ( DEV_FULL.NUM_CYLINDERS,  6, 53, 4 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.NUM_TRACKS,     7, 53, 4 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.NUM_SECTORS,    8, 53, 4 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.CLUSTER_SIZE,   9, 53, 4 )
C
C         Output max files, operations completed, and default buffer size.
C
          STATUS = OUTPUT_INTEGER ( DEV_FULL.MAX_FILES,      6, 73, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.OPS_COMPLETE,   7, 73, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.DEF_BUFF_SZ,    8, 73, 8 )
C
C         Output number of split ios, window turns, window hits,
C         file header cache hits, file header cache misses, and
C         number of currently open files.
C
          STATUS = OUTPUT_INTEGER ( DEV_FULL.SPLIT,          12, 12, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.TURN,           13, 12, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.HIT,            14, 12, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.FILHDR_HIT,     15, 12, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.FILHDR_MISS,    16, 12, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.OPEN,           17, 12, 8 )
C
C         Output number of direct ios, directory LRU hits, directory
C         LRU misses, directory data block hits, and directory data
C         block misses.
C
          STATUS = OUTPUT_INTEGER ( DEV_FULL.DIRIO,          12, 33, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.DIRHIT,         13, 33, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.DIRMISS,        14, 33, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.DIRDATA_HIT,    15, 33, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.DIRDATA_MISS,   16, 33, 8 )
C
C         Output total number of file opens, file ID cache hits,
C         file ID cache misses, storage bit map cache hits, and
C         storage bit map cache misses.
C
          STATUS = OUTPUT_INTEGER ( DEV_FULL.OPENS,          12, 54, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.FIDHIT,         13, 54, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.FIDMISS,        14, 54, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.STORAGMAP_HIT,  15, 54, 8 )
          STATUS = OUTPUT_INTEGER ( DEV_FULL.STORAGMAP_MISS,16, 54, 8 )
```

```
C
C       Output total erase ios, extent cache hits, extent cache misses,
C       quota cache hits, and quota cache misses.
C
        STATUS = OUTPUT_INTEGER ( DEV_FULL.ERASEIO,       12, 73, 8 )
        STATUS = OUTPUT_INTEGER ( DEV_FULL.EXTHIT,        13, 73, 8 )
        STATUS = OUTPUT_INTEGER ( DEV_FULL.EXTMISS,       14, 73, 8 )
        STATUS = OUTPUT_INTEGER ( DEV_FULL.QUOHIT,        15, 73, 8 )
        STATUS = OUTPUT_INTEGER ( DEV_FULL.QUOMISS,       16, 73, 8 )
C
C       Output fragmentation factor and fragmentation statistics.
C
        STATUS = OUTPUT_REAL ( DEV_FULL.FRAG_FACTOR, 19, 75, 6, 3 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(1), 20, 8, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(2), 21, 8, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(3), 22, 8, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(4), 23, 8, 5 )

        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(5), 20, 28, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(6), 21, 28, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(7), 22, 28, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(8), 23, 28, 5 )

        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE( 9), 20, 51, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(10), 21, 51, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(11), 22, 51, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(12), 23, 51, 5 )

        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(13), 20, 76, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(14), 21, 76, 5 )
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(15), 22, 76, 5 )
        DEV_FULL.FRAG_USAGE(16) = DEV_FULL.FRAG_USAGE(16) +  ! sum all > 100000
     1                            DEV_FULL.FRAG_USAGE(17) +
     2                            DEV_FULL.FRAG_USAGE(18)
        STATUS = OUTPUT_INTEGER( DEV_FULL.FRAG_USAGE(16), 23, 76, 5 )
C
      END IF
C
C       Position the cursor to the top of the display.
C       Turn off batching of the input.  This causes the SMG buffers
C       to be physically displayed to the terminal.
C
      STATUS = SMG$SET_CURSOR_ABS ( DDID(FORM), 1, 1 )
      STATUS = SMG$END_DISPLAY_UPDATE ( DDID(FORM) )

      UPDATE_DYNAMIC = 1 !!last non-successful status or ss$_normal.  NOT 1!!!!

      RETURN
      END
```

A.45 UPDATE_SCREEN

```
      INTEGER*4      FUNCTION UPDATE_SCREEN ( FORM,
      1                                       NUM_DEVICES, DEV_BRIEF,
      2                                       DEV_FULL,
      3                                       REDRAW_FLAG, TOKEN,
      4                                       DEV_NUM )
C
C     name:          Update Screen
C
C     abstract:      Update the screen the operator is currently viewing.
C
C     history:       11/01/87---pdl by RGD
C
C     includes:
C
      INCLUDE        'MS_INC:DISK_ANAL_CONST.INC'     ! constant definitions
      INCLUDE        'MS_INC:DEV_ATT.INC'
                     ! device attribute structure definition
      INCLUDE        'MS_INC:TOKEN_DEFN.INC'          ! token definitions
C
C     arguments:
C
      INTEGER*2      FORM                             ! current screen id
      INTEGER*2      NUM_DEVICES
                     ! actual number of devices being displayed
      RECORD         /DEV_BRIEF_ATT/ DEV_BRIEF( MAX_DEVICES )
      RECORD         /DEV_FULL_ATT/  DEV_FULL
      LOGICAL*2      REDRAW_FLAG                      ! redraw all of screen?
      RECORD         /TOKEN/ TOKEN                    ! token is current field marker
      INTEGER*2      DEV_NUM                          ! device index for form 2
C
C     globals:
C


C
C     locals:
C
      INTEGER*4      STATUS                           ! return status
C
C     functions:
C
      INTEGER*4      REDRAW_SCREEN                    ! redraws static portion of screen
      INTEGER*4      UPDATE_DYNAMIC
                     ! redraws dynamic portion of screen
C
C.....................................
C
C
C     If the redraw flag is true, then the screen has been
C     destroyed.  We must redraw the screen for the current form.
C
      IF ( REDRAW_FLAG ) THEN
         STATUS = REDRAW_SCREEN ( FORM, NUM_DEVICES )
```

```
            IF ( .NOT. STATUS ) THEN
               UPDATE_SCREEN = STATUS
               RETURN
            END IF
            REDRAW_FLAG = .FALSE.
         END IF
C
C        Update the dynamic portion of the screen.  These are the values
C        that were retrieved from GET_DEVICE_INFO calls.
C
         STATUS = UPDATE_DYNAMIC ( FORM, NUM_DEVICES,
         1                         DEV_BRIEF(1), DEV_FULL,
         2                         TOKEN, DEV_NUM )
         IF ( .NOT. STATUS ) THEN
            UPDATE_SCREEN = STATUS
            RETURN
         END IF

         UPDATE_SCREEN = STATUS

         RETURN
         END
```

A.46 UPDATE_TOKEN

```
         INTEGER*4       FUNCTION UPDATE_TOKEN ( TOKEN, F_KEY,
         1                                       FORM, NUM_DEVICES )
C
C        name:           Update Function Key
C
C        abstract:       Update the location of the function key on the display.
C
C        history:        01/03/88--pdl by RGD
C
C        includes:
C
         INCLUDE         'MS_INC:TOKEN_DEFN.INC'          ! token definitions
         INCLUDE         'MS_INC:DISK_ANAL_IO_DEFN.INC'   ! I/O definitions
         INCLUDE         'MS_INC:FORM2_TOKEN_POS.INC'
                         ! token position definitions for form 2
         INCLUDE         '($SMGDEF)'                      ! SMG definitions
C
C        arguments:
C
         RECORD          /TOKEN/ TOKEN                    ! input token
         INTEGER*2       FORM                             ! current form being displayed
         INTEGER*2       F_KEY                            ! input function key
         INTEGER*2       NUM_DEVICES
                         ! number of devices being displayed
C
C        globals:
```

```
C
C
C
C       locals:
C
        INTEGER*4       STATUS                          ! function return status
        CHARACTER*1     BLANK_TOKEN                     ! a blank token
        DATA            BLANK_TOKEN / ' ' /
C
C       functions:
C
        INTEGER*4       SMG$SET_CURSOR_ABS              ! smg routines
        INTEGER*4       SMG$PUT_CHARS
        INTEGER*4       SMG$BEGIN_DISPLAY_UPDATE
        INTEGER*4       SMG$END_DISPLAY_UPDATE
C
C.....................................
C
C       Turn on batching of output to display.
C
        STATUS = SMG$BEGIN_DISPLAY_UPDATE ( DDID(FORM) )
C
C       Erase current position of token on the screen.
C
        STATUS = SMG$PUT_CHARS ( DDID(FORM), %DESCR(BLANK_TOKEN),
        1                        TOKEN.ROW, TOKEN.COL )
C
C       Move token to its new position.
C
        IF ( FORM .EQ. 1 ) THEN
           IF ( F_KEY .EQ. SMG$K_TRM_UP ) THEN                  ! prev field
              IF ( TOKEN.FIELD .GT. 1 ) THEN
                 TOKEN.FIELD = TOKEN.FIELD - 1
                 TOKEN.ROW = TOKEN.ROW - 1
              ELSE ! wrap
                 TOKEN.FIELD = NUM_DEVICES
                 TOKEN.ROW = NUM_DEVICES + 6
              END IF
           ELSE IF ( F_KEY .EQ. SMG$K_TRM_DOWN ) THEN           ! next field
              IF ( TOKEN.FIELD .LT. NUM_DEVICES ) THEN
                 TOKEN.FIELD = TOKEN.FIELD + 1
                 TOKEN.ROW = TOKEN.ROW + 1
              ELSE ! wrap
                 TOKEN.FIELD = 1
                 TOKEN.ROW = 7
              END IF
           END IF
        ELSE ! form = 2
           IF ( F_KEY .EQ. SMG$K_TRM_UP ) THEN                  ! prev field
              TOKEN.FIELD = TOKEN.FIELD - 1
              IF ( TOKEN.FIELD .LT. 1 ) TOKEN.FIELD = MAX_FIELDS! wrap
           ELSE IF ( F_KEY .EQ. SMG$K_TRM_DOWN ) THEN           ! next field
              TOKEN.FIELD = TOKEN.FIELD + 1
              IF ( TOKEN.FIELD .GT. MAX_FIELDS ) TOKEN.FIELD = 1! wrap
```

```
            END IF
            TOKEN.ROW = FORM2_ROW( TOKEN.FIELD )
            TOKEN.COL = FORM2_COL( TOKEN.FIELD )
         END IF
C
C        Output token at its new position.
C
         STATUS = SMG$PUT_CHARS ( DDID(FORM), %DESCR(TOKEN.CHR),
         1                        TOKEN.ROW, TOKEN.COL )
C
C        Turn batching off such that screen update will occur.
C
         STATUS = SMG$END_DISPLAY_UPDATE ( DDID(FORM) )
C
C        Set function return status.
C
         UPDATE_TOKEN = 1 !!last nonsuccessful status
         RETURN
         END
```

APPENDIX B

DISKMGR INCLUDE FILE LISTINGS

This appendix contains a complete listing of all the DISKMGR include files. These files are listed in alphabetical order.

B.1 COMMAND_DEFN

```
C
C       name:           Command Definitions
C
C       abstract:       Defines command array for use by DISKMGR
C
C       history:        01/19/88---pdl by RGD
C
        CHARACTER*80    COMMAND ( MAX_FIELDS + 1 )

        COMMON          /DISKMGR_COMMAND/ COMMAND
```

B.2 DEV_ATT

```
C
C       name:           Device Attributes
C
C       abstract:       The first structure of elements
C                       are used on form 1.  The entire set of
C                       device attributes are displayed on form 2.
C                       Return lengths are required for all character
C                       string values returned.  Return lengths of
C                       integer values are discarded by being returned
C                       into a "dummy" value.  Two structures are used
C                       here since only one copy of the full attributes
C                       are required at any given time.  However, if
```

```
C                   form 1 is current, then many copies of device
C                   brief attributes are required.
C
C       history:    11/01/87---pdl by RGD
C
        STRUCTURE   /DEV_BRIEF_ATT/
          CHARACTER*64 DEVICE_NAME                ! device name
          INTEGER*4    DEV_NAM_LEN                ! device name length
          CHARACTER*12 VOLUME_LABEL               ! volume name
          INTEGER*4    DEV_VOL_NAME_LEN           ! volume name length
          INTEGER*4    ERROR_COUNT                ! error count
          INTEGER*4    MAX_BLOCKS                 ! maximum blocks on device
          INTEGER*4    FREE_BLOCKS                ! free block count
          INTEGER*4    TRANS_COUNT                ! transaction count
          INTEGER*4    MOUNT_COUNT                ! mount count
        END STRUCTURE

        STRUCTURE   /UIC/
          UNION
            MAP
              INTEGER*2    GROUP
              INTEGER*2    MEMBER
            END MAP
            MAP
              INTEGER*4    UIC
            END MAP
          END UNION
        END STRUCTURE


        STRUCTURE   /DEV_FULL_ATT/
          INTEGER*4    MEDNAM_LEN                 ! length of type of disk name
          CHARACTER*10 MEDIA_NAME                 ! type of disk
          INTEGER*4    ACP_TYPE                   ! level 1 or level 2?
          INTEGER*4    MOUNTED                    ! is disk mounte?
          INTEGER*4    PID                        ! owner process id
          RECORD       /UIC/ OWNUIC               ! owner uic
          INTEGER*4    PROT                       ! device protection
          INTEGER*4    CLUSTER_SIZE               ! cluster factor size
          INTEGER*4    NUM_CYLINDERS              ! number of cylinders
          INTEGER*4    NUM_TRACKS                 ! number of tracks
          INTEGER*4    NUM_SECTORS                ! number of sectors
          INTEGER*4    MAX_FILES                  ! maximum number of files
          INTEGER*4    OPS_COMPLETE               ! operations count
          INTEGER*4    DEF_BUFF_SZ                ! device buffer size
          INTEGER*4    REF_COUNT                  ! reference count
          INTEGER*4    SPLIT                      ! # of split io transfers
          INTEGER*4    DIRIO                      ! # direct io operations
          INTEGER*4    OPENS                      ! # file opens
          INTEGER*4    ERASEIO                    ! # erase $QIOs issued
          INTEGER*4    TURN                       ! # window turns
          INTEGER*4    HIT                        ! # transfers not requiring window turns
          INTEGER*4    DIRHIT                     ! # directory LRU hits
          INTEGER*4    DIRMISS                    ! # directory LRU misses
```

```
        INTEGER*4   FIDHIT              ! # file id cache hits
        INTEGER*4   FIDMISS             ! # file id cache misses
        INTEGER*4   EXTHIT              ! # extent cache hits
        INTEGER*4   EXTMISS             ! # extent cache misses
        INTEGER*4   FILHDR_HIT          ! # file header cache hits
        INTEGER*4   FILHDR_MISS         ! # file header cache misses
        INTEGER*4   DIRDATA_HIT         ! # directory data block hits
        INTEGER*4   DIRDATA_MISS        ! # directory data block misses
        INTEGER*4   STORAGMAP_HIT       ! # storage bit map cache hits
        INTEGER*4   STORAGMAP_MISS      ! # storage bit map cache misses
        INTEGER*4   QUOHIT              ! # quota cache hits
        INTEGER*4   QUOMISS             ! # quota cache misses
        INTEGER*4   OPEN                ! # files currently opened
        INTEGER*4   FRAG_USAGE(-1:19)   ! file fragment distribution
        REAL*4      FRAG_FACTOR         ! fragmentation factor
    END STRUCTURE
```

B.3 DISK_ANAL_CONST

```
C
C       name:           Disk Analysis Constant Definitions
C
C       abstract:       Constant definitions for disk analysis program.
C
C       history:        11/01/87-pdl by RGD
C
        INTEGER*2       MAX_DEVICES             ! max num devices to be displayed

        PARAMETER       ( MAX_DEVICES = 16 )
```

B.4 DISK_ANAL_INIT

```
C
C       name:           Disk Analysis Initialization definitions
C
C       abstract:       Initializations for disk analysis program
C
C       history:        11/01/87---pdl by RGD
C
C       REDRAW_FLAG     - .TRUE. ==> redraw background and static variables of
C                                    current form
C                         .FALSE.==> background does not need to be redrawn;
C                                    only redraw the part of screen that is dynamic
C
C       FIRST_DEVICE    - (1)    ==> for form=1, this is the first device to be displayed
C                       - (2)    ==> for form=2, this is the first, and only,
C                                                device to be displayed
C
C       LAST_DEVICE     - (1)    ==> for form=1, this is the last device to be displayed.
C                       - (2)    ==> for form=2, this is the last, and only,
```

```
C                                                   device to be displayed
C
      LOGICAL*2       REDRAW_FLAG
      INTEGER*2       FIRST_DEVICE(2)
      INTEGER*2       LAST_DEVICE(2)

      DATA            REDRAW_FLAG  / .TRUE. /
      DATA            FIRST_DEVICE / 1, 1 /
      DATA            LAST_DEVICE  / 0, 1 /          ! default for /FULL qualifier
```

B.5 DISK_ANAL_IO_DEFN

```
C
C       name:         Disk Analysis IO Definitions
C
C       abstract:     Define default IO values for use by
C                     disk analysis input/output routines.
C
C       history:      11/01/87--pdl by RGD
C
      INTEGER*4       PBID                           ! pasteboard id
      INTEGER*4       DDID(5)                        ! data display ids
      INTEGER*4       ROWS                           ! number rows of physical
      INTEGER*4       COLS                           ! number cols of physical device
      INTEGER*4       VKID                           ! virtual keyboard id

      COMMON /IO_DEFS/ PBID, DDID, ROWS, COLS, VKID
```

B.6 FORM2_TOKEN_POS

```
C
C       name:         Form 2 token position
C
C       abstract:     Token position definitions for form 2.
C
C       history:      01/13/88--pdl by RGD
C
      INTEGER*2       MAX_FIELDS
      PARAMETER       ( MAX_FIELDS = 46 )
      INTEGER*2       FORM2_ROW ( MAX_FIELDS ) ! physical cursor positions
      INTEGER*2       FORM2_COL ( MAX_FIELDS )

      DATA            FORM2_ROW /  4,  4,  4,  4,
     1                             4,  5,  5,  5,
     2                             6,  6,  6,  6,
     3                             7,  7,  7,  7,
     4                             8,  8,  8,  8,
     5                             9,  9,  9, 12,
     6                            12, 12, 12, 13,
     7                            13, 13, 13, 14,
```

```
     8                         14, 14, 14, 15,
     9                         15, 15, 15, 16,
     1                         16, 16, 16, 17,
     2                         19, 19 /

     DATA            FORM2_COL /  1, 15, 29, 35,
     1                          54,  1, 22, 41,
     2                           1, 22, 41, 60,
     3                           1, 22, 41, 60,
     4                           1, 22, 41, 60,
     5                           1, 22, 41,  1,
     6                          21, 42, 63,  1,
     7                          21, 42, 63,  1,
     8                          21, 42, 63,  1,
     9                          21, 42, 63,  1,
     1                          21, 42, 63,  1,
     2                           1, 53 /
```

B.7 FORM_BCKGRND_DEFN

```
C
C       name:           Background Definitions.
C
C       abstract:       Background definitions for the forms used
C                       for displaying the device information.
C
C       history:        01/01/88--pdl by RGD
C
C
C       form = 1
C
        CHARACTER*80    BLANKLINE
        CHARACTER*80    SUBHEADER(3)
        CHARACTER*80    TOTAL(2)

        DATA        BLANKLINE /'
        1                                          '/

        DATA        SUBHEADER /'  DEVICE         VOLUME     ERROR    M
        1AX #    FREE #   PERCENT   TRANS   MOUNT',
        2                   '     NAME          LABEL      COUNT    B
        3LOCKS   BLOCKS     USED    COUNT   COUNT',
        4                   '_____________ ____________ _____   _
        5_____   ______    _____   _____   _____'/

        DATA            TOTAL /'                                    _____   _
        1_____   ______    _____   _____   _____',
        2                   '                    TOTALS:
        3                                         '/
C
C       form = 2
C
```

```
CHARACTER*80    LINE(20)                ! line definitions for form 2
DATA            LINE /'
1                                       ',
3                       ' OWNER PID:       UIC: [999,999]
3 PROT: (S:     ,O:     ,G:     ,W:     )  ',
5                       ' MAX BLKS:        MOUNT CNT:
5 #CYLINDERS:          MAX FILES:          ',
6                       ' FREE BLKS:       TRANS CNT:
6 #TRACKS:             OPERATIONS:         ',
7                       ' BLKS USED:       REF CNT:
7 #SECTORS:            DEF.BUFF.SZ:        ',
8                       ' PERC USED:       ERROR CNT:
8 CLUSTER SZ:                             ',
9                       '
9                                         ',
1                       ' FILE PRIMITIVE & SYSTEM CACHING STATISTI
1CS ( counts since boot-up )              ',
2                       ' SPLIT IOS:       DIRECT IOS:
2 FILE OPENS:           ERASEIOS:          ',
3                       ' WIND TURN:       DIR HITS:
3 FID HITS:             EXTNTHIT:          ',
4                       ' WIND HITS:       DIR MISS:
4 FID MISS:             EXTNTMIS:          ',
5                       ' FILHDRHIT:       DIRDATAHIT:
5 BITMAPHIT:            QUOTAHIT:          ',
6                       ' FILHDRMIS:       DIRDATAMIS:
6 BITMAPMISS:           QUOTAMIS:          ',
7                       ' #FILESOPN:
7                                         ',
9                       '
9                                         ',
1                       ' FRAGMENTATION STATISTICS ( free space o
1nly )          FRAGMENTATION FACTOR:     ',
2                       '  1- 3:           26- 50:
2301-  500:             10001- 25000:     ',
3                       '  4- 6:           51-100:
3501- 1000:             25001- 50000:     ',
4                       '  7-10:           101-200:          1
4001- 5000:             50001-100000:     ',
5                       ' 11-25:           201-300:          5
5001-10000:                 > 100000:     ' /
```

B.8 GETDVI_CONST

```
C
C       name:           Get Device Information Constant Definitions
C
C       abstract:       Define constants used to obtain information
C                       specific to a device.
C
C       history:        11/01/87-pdl by RGD
C
```

```
      INTEGER*2       NUM_ITEMS(2)            ! num items for each form

      DATA            NUM_ITEMS / 7, MAX_ITEMS /
```

B.9 HEADER_BCKGRND_DEFN

```
C
C       name:         Header background definitions.
C
C       abstract:     Header background definitions for the forms used
C                     for displaying the device information.
C
C       history:      01/01/88---pdl by RGD
C
        CHARACTER*80  TITLE
        CHARACTER*80  DATETIME
        CHARACTER*80  BLANKLINE

        DATA          TITLE /'                                SVT/UTSI/RGD
        1 DISK MANAGER                              '/

        DATA        DATETIME /'                                DD-MMM-YYYY
        1 HH:MM:SS.CC                               '/

        DATA       BLANKLINE /'
        1                                           '/
```

B.10 HELP_FIELD_DEFN

```
C
C       name:         Help field definitions
C
C       abstract:     Defines help field values for use by
C                     routine FIELD_HELP.
C
C       history:      01/16/88---pdl by RGD
C
        CHARACTER*22  FIELD_NAME( MAX_FIELDS+1 )

        DATA          FIELD_NAME / 'FORM 1              ',
        1                          'DEVICE_NAME         ',
        2                          'VOLUME_NAME         ',
        3                          'MEDIA_TYPE          ',
        4                          'ACP_TYPE_CODE       ',
        5                          'DEVICE_CHARS        ',
        6                          'OWNER_PID           ',
        7                          'OWNER_UIC           ',
        8                          'DEVICE_PROT         ',
        9                          'MAXIMUM_BLOCKS      ',
        1                          'MOUNT_COUNT         ',
```

```
2                       'NUM_CYLINDERS      ',
3                       'MAXIMUM_FILES      ',
4                       'FREE_BLOCKS        ',
5                       'TRANS_CNT          ',
6                       'NUM_TRACKS         ',
7                       'OPERATIONS         ',
8                       'BLOCKS_USED        ',
9                       'REFERENCE_COUNT    ',
1                       'NUM_SECTORS        ',
2                       'BUFFER_SIZE        ',
3                       'PERCENT_USED       ',
4                       'ERROR_COUNT        ',
5                       'CLUSTER_SIZE       ',
4                       'SPLIT_IOS          ',
4                       'DIRECT_IOS         ',
4                       'FILE_OPENS         ',
4                       'ERASE_IOS          ',
4                       'WINDOW_TURNS       ',
4                       'DIRECTORY_HITS     ',
4                       'FILE_ID_HITS       ',
4                       'EXTENT_HITS        ',
4                       'WINDOW_HITS        ',
4                       'DIRECTORY_MISS     ',
4                       'FILE_ID_MISSES     ',
4                       'EXTENT_MISSES      ',
4                       'FILE_HDR_HITS      ',
4                       'DIR_DATA_HITS      ',
4                       'BIT_MAP_HITS       ',
4                       'QUOTA_HITS         ',
4                       'FILE_HDR_MISSES    ',
4                       'DIR_DATA_MISSES    ',
4                       'BIT_MAP_MISSES     ',
4                       'QUOTA_MISSES       ',
4                       'NUM_FILES_OPEN     ',
4                       'FRAG_STATS         ',
4                       'FRAG_FACTOR        ' /
```

B.11 ITMLST_STRUCT_DEFN

```
C
C       name:           Item List Structure Include
C
C       abstract:       Use an item list to describe the information
C                       to be returned from the GETDVI system service.
C
C       history:        11/01/87---pdl by RGD
C
        STRUCTURE       /ITMLST/
          UNION
            MAP
              INTEGER*2     BUFFER_LENGTH   ! length (in bytes) of buffer
              INTEGER*2     ITEM_CODE       ! defined by $DVIDEF macro
```

```
            INTEGER*4       BUFFER_ADDRESS  ! address for storing information
            INTEGER*4       RETURN_LENGTH   ! actual length written to buffer
          END MAP
          MAP
            INTEGER*4       END_LIST        ! end of itemlist
          END MAP
        END UNION
      END STRUCTURE
C
C     Define the maximum number of items that need to be retrieved.
C
      INTEGER*2       MAX_ITEMS
      PARAMETER       ( MAX_ITEMS = 21 )
```

B.12 LIBRARY_DEFN

```
C
C     name:           Library Definitions
C
C     abstract:       Defines all the constants necessary for
C                     for accessing the DISKMGR.HLB help library.
C                     These are the data structures used by the
C                     LBR$ routines
C
C     history:        02/01/88---pdl by RGD
C
      INTEGER*4       LIBRARY_INDEX                   ! library control index
      INTEGER*4       FUNCTION                        ! the function performed
      INTEGER*4       TYPE                            ! the library type
      CHARACTER*19    LIBRARY_NAME                    ! library to be searched
      DATA            LIBRARY_NAME / 'MS_HELP:DISKMGR.HLB' /

      COMMON          / LIBRARY / LIBRARY_INDEX
```

B.13 TOKEN_DEFN

```
C
C     name:           Token Definitions
C
C     abstract:       Defines the data structure known as TOKEN
C
C     history:        01/03/88---pdl by RGD
C
      STRUCTURE       /TOKEN/
        INTEGER*4     ROW                             ! absolute row offset in display
        INTEGER*4     COL                             ! absolute col offset in display
        INTEGER*4     FIELD                           ! current field of display
        CHARACTER*1   CHR                             ! token character
      END STRUCTURE
```

APPENDIX C

DISKMGR FIELDS AND COMMANDS

This appendix contains a cross reference of field names and the VMS commands executed for each field. This table defines DISKMGR's function when the execute command function key is active.

FIELD NAME	VMS COMMAND
FORM 1	SHOW DEVICE D
DEVICE NAME	SHOW DEVICE D
VOLUME NAME	SHOW DEVICE D
MEDIA TYPE	SHOW DEVICE D
ACP TYPE CODE	SHOW DEVICE D
DEVICE CHARS	SHOW DEVICE D
OWNER PID	SHOW DEVICE D
OWNER UIC	SHOW DEVICE D
DEVICE PROT	SHOW DEVICE D
MAXIMUM BLOCKS	SHOW DEVICE D
MOUNT COUNT	SHOW DEVICE D
NUM CYLINDERS	SHOW DEVICE D
MAXIMUM FILES	SHOW DEVICE D
FREE BLOCKS	SHOW DEVICE D
TRANS CNT	SHOW DEVICE <disk>/FILES
NUM TRACKS	SHOW DEVICE D
OPERATIONS	SHOW DEVICE D
BLOCKS USED	SHOW DEVICE D
REFERENCE COUNT	SHOW DEVICE <disk>/FILES
NUM SECTORS	SHOW DEVICE D
BUFFER SIZE	SHOW DEVICE D
PERCENT USED	SHOW DEVICE D
ERROR COUNT	ANALYZE/ERROR/SUMMARY/INCLUDE=<device>
CLUSTER SIZE	SHOW DEVICE D

FIELD NAME	VMS COMMAND
SPLIT_IOS	MONITOR DISK/ITEM=ALL
DIRECT_IOS	MONITOR PROCESS/TOPDIO
FILE_OPENS	MONITOR FCP
ERASE_IOS	MONITOR FCP
WINDOW_TURNS	MONITOR FCP
DIRECTORY_HITS	MONITOR FILE_SYSTEM_CACHE
FILE_ID_HITS	MONITOR FILE_SYSTEM_CACHE
EXTENT_HITS	MONITOR FILE_SYSTEM_CACHE
WINDOW_HITS	MONITOR FCP
DIRECTORY_MISS	MONITOR FILE_SYSTEM_CACHE
FILE_ID_MISSES	MONITOR FILE_SYSTEM_CACHE
EXTENT_MISSES	MONITOR FILE_SYSTEM_CACHE
FILE_HDR_HITS	MONITOR FILE_SYSTEM_CACHE
DIR_DATA_HITS	MONITOR FILE_SYSTEM_CACHE
BIT_MAP_HITS	MONITOR FILE_SYSTEM_CACHE
QUOTA_HITS	MONITOR FILE_SYSTEM_CACHE
FILE_HDR_MISSES	MONITOR FILE_SYSTEM_CACHE
DIR_DATA_MISSES	MONITOR FILE_SYSTEM_CACHE
BIT_MAP_MISSES	MONITOR FILE_SYSTEM_CACHE
QUOTA_MISSES	MONITOR FILE_SYSTEM_CACHE
NUM_FILES_OPEN	MONITOR FCP
FRAG_STATS	SHOW DEVICE D
FRAG_FACTOR	SHOW DEVICE D

VITA

Randy G. Dotson was born in Lafayette, Tennessee on August 25, 1963. He graduated from Pulaski County High School in Somerset, Kentucky, in June 1981. The following August he entered Eastern Kentucky University, and in August 1984 he received a Bachelor of Science degree in Computer Science with a second major in Mathematics. Immediately following graduation he accepted a system manager position with Sverdrup Technology, Inc. in Tullahoma, Tennessee. In September of the same year, he began study toward a Master's degree at the University of Tennessee Space Institute. This degree was awarded in June 1988.