SDOS™

6800/6809 DISK OPERATING SYSTEM

REFERENCE MANUAL

SOFTWARE DYNAMICS
2111 W. Crescent, Suite G, ▲ Anaheim, CA 92801

S D O S 1 . 1

U S E R ' S

M A N U A L

3rd Printing

NOTICE

This manual describes Software Dynamics Operating System (SDOS) Version 1.1. Software Dynamics has carefully checked the information given in this manual, and it is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies.

SD reserves the right to change the specifications without notice.

This manual is a major revision of the SDOS 1.Ø manual. SDOS 1.Ø users are recommended to read it completely.

WARNINGs about dangerous operations can be found in the index. We suggest you review them all before using the system seriously.

READ ME FIRST!

Hello. We know you are anxious to begin use of your new computer system and software. However, use of a new tool is always a little bit dangerous to the uninitiated, so we recommend you follow the steps in this section exactly BEFORE YOU ATTEMPT ANYTHING ELSE. The biggest "danger" you face is accidental destruction (erasure, deletion of critical system "files", warpage, etc.) of your system disk before you have made a duplicate of it; this will leave your computer helpless and you frustrated. This section shows you how to boot the computer, enter your SD Registration code, and make a Backup copy of your system disk. Once your system disk has been safely duplicated, destruction of the system disk isn't nearly so bad; you merely use the duplicate!

The section on BOOTING (see Table of Contents) describes how to start up your computer. Read that section now, and come back here when SDOS first says, "Hello.....", or you get ".TIME" printed out on your screen.

Now your system is running SDOS. If ".TIME" is printed, skip forward to "BACKING UP THE BOOT DISK". Otherwise, you have "Hello..." on your screen, and SDOS wants you to enter a Registration code. The directions should be straightforward.

HANDLING THE REGISTRATION CODE

The computer will show you a 16 digit serial number (composed of letters or digits). You must give this number, along with your (organization's) name to Software Dynamics. SD will return a 16 digit (likewise composed of letters or digits) number which is a "registration code". If you call SD with this information, SD will give the registration code by phone; if you mail it, SD will mail back your registration code. You must enter both your (organization) name (exactly as you gave it to SD) and the registration code (exactly as SD gave it to you) into the computer when it asks for them. If you don't enter them exactly, SDOS will refuse to accept the registration code. Push <CR> (the RETURN key) after entering your name and after entering the registration code. If you make a mistake while typing, push <DELETE> or <RUBOUT> once for each mistyped character.

Example:

```
Hello, ..............
..................... text about registration ............
..........................................................

This is CPU Serial Number FF0000000000001
Enter Purchaser Name: Ferd Automotive, Inc.
The following message will appear at 'boot' time:

This copy licensed for use only by Ferd Automotive, Inc.
only on the single computer with CPU serial number FF0000000000001

Enter the SD registration code between the brackets,
followed by <RETURN> key, to validate name exactly as shown;
Enter <RETURN> if it is NOT right.
SD registration code:>0123456789ABCDEF<

Proper registration code entered. Your name is now frozen
Don't forget to IMMEDIATELY make a Backup of your System disk!
```

BACKING UP THE BOOT DISK

Now your screen shows

```
.TIME
```

Press the Return key (you can learn about the TIME command later).

Now we will demonstrate how to make a backup copy of your boot disk. Once having made the backup, we suggest storing the original boot disk in a safe place away from the computer. Use the duplicate as a "working boot disk"; also make a copy of the duplicate so you have two boot disks. The idea is to keep the master disk away from the machine, ESPECIALLY when recovering from a disaster.

How one makes a backup copy of the system disk depends on the configuration of your computer hardware. There are three common configurations:

1) Computer with two identical floppy disk drives
2) Computer with one floppy disk and one Winchester disk drive
3) Computer with only a single floppy disk

All three of these require a freshly formatted disk with the same format as the boot disk on which the duplicate copy will be placed. If you don't know what this means, ask someone technical. A special, manufacturer-specific formatter program (named FORMAT) generally comes with SDOS (although it is not part of SDOS). Refer to the manufacturer's documentation for how to use it. Then use the formatter program to make at least one freshly fomatted diskette. Depending on the manufacturer, you may have to re-boot after formatting.

To make a backup with two identical disk drives, named DØ: and D1:, where DØ: is the name of the Boot disk drive, first place a fresh diskette in D1:, and type:

```
.SDOSDISKBACKUP DØ: TO D1:
*** SDOSDISKBACKUP V1.1g ***
Writing on the DISK device can damage the file structure.
Are you sure you want to write on the DISK device? YES
Copying DØ: to D1:
```

When the '.' prompt returns, D1: is an exact copy of DØ:. You can take the diskettes out of the computer after you type:

```
.DISMOUNT DØ:
.DISMOUNT D1:
```

To make a backup with a floppy named DØ: and a Winchester named WDØ:, type:

```
.SDOSDISKBACKUP DØ: TO WDØ:DISK.IMAGE
```

Now type:

```
.DISMOUNT DØ:
```

Place a fresh blank formatted floppy into DØ:, and then type:

```
.SDOSDISKBACKUP WDØ:DISK.IMAGE TO DØ:
*** SDOSDISKBACKUP V1.1g ***
Writing on the DISK device can damage the file structure.
Are you sure you want to write on the DISK device? YES
Copying WDØ:DISKIMAGE to DØ:
```

When the '.' prompt returns, DØ: is an exact copy of the original disk. Before removing the floppy from the computer, type:

```
.DISMOUNT DØ:
```

To make a backup with only one disk drive, you need to use the SDCOPY (Single-Disk Copy) program. First, go read the section on SDCOPY, then come back here. Type:

```
.SDCOPY
Single Disk Copy v1.xx
Name of Source File:  DØ:
Name of Target File:  DØ:
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
...
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
COPY COMPLETE: Insert System Disk, hit <RETURN>
.
```

When the '.' prompt returns, the "target" diskette is an exact copy of the original. There is no need to DISMOUNT the target floppy as SDCOPY has done this automatically.

INTRODUCTION

This manual describes SDOS 1.1, a 6800/6809 disk operating system. The documentation for SDOS has several sections:

1) Features, being a very short description of the major facilities of SDOS.

2) Concepts, section where simple definitions of terms used throughout the rest of the manual are given. Novices should read this before going on to Section 3.

3) Operator's Guide. This describes how to initiate execution of application and support programs in detail. Novices should read this before attempting to use the software.

4) SDOS Interface to user assembly programs. This section describes system calls and control block formats.

5) SDOS architecture. This section describes the structure of SDOS and the file system.

6) I/O package. This section describes the I/O package used to interface SDOS to peripheral devices, and how to modify it.

Parts 1, 2 and 3 are included in this manual.

Part 4 is covered in the "SDOS Application Programmer's Guide."

Parts 5 and 6 are published separately as "SDOS Systems Implementer's Guide".

To use the full power of SDOS, all of these manuals are needed. For everyday use, this manual should be sufficient.

This manual is designed to be read from front to back. We STRONGLY suggest you do so at least once before attempting any work with the software.

SDOS USER'S MANUAL
SECTION I: INTRODUCTION

SDOS FEATURES

SDOS is a family of 6800/6809 microprocessor-based Operating Systems.

The family includes single-user, multi-user and network operating systems. This manual describes the single and multi-user versions of SDOS.

SDOS provides many features which ease and simplify the construction and execution of application programs. These features include:

1) Device independence: the ability to treat all devices the same way.

2) Named files: users need only remember assigned names for the programs they wish to use.

3) Automatic disk file management: SDOS allocates and frees disk space automatically as needed by write requests. Space management is dynamic, but optimized for quick access whenever possible.

4) Multiple and mixed disk device support: both mini-floppies and 60 megabyte storage modules can be attached to the same system running SDOS.

5) Error trapping and automatic reporting: most errors are printed on the console in English text instead of cryptic numbers (A HELP command converts the remaining cryptic numbers to English text). Application programs can capture and attempt recovery from virtually any error.

6) Hashed disk directory with automatic expansion: hashing ensures quick look-up of file names; automatic directory expansion means that disk space is the only limit to the number of files on a disk.

7) Sequential and randomly addressable (to the byte) disk files: any file can be processed both sequentially and randomly. Read-ahead improves performance on sequential reads. The SDOS file structure ensures that no more than two disk reads are necessary to randomly access a file; buffering in SDOS normally trims this to a single disk read, even for files scattered over the entire disk.

8) A command interpreter: a package which contains many useful utilities for listing files, copying, etc., is automatically loaded when application programs stop running.

9) Latency and spiral tuning: to allow sequential read optimization.

10) Many utility programs: to aid initializing, copying, and repairing SDOS disk file systems.

11) Command files: allow sequences of keyboard commands to be stored and later executed. Conditional execution allows recovery from processing errors.

12) Easy addition of new peripheral drivers.

13) Interrupt-driven I/O: enhances system throughput.

14) Encrypted program feature: ensures applications run only on the serial numbered processor for which they were intended.

NOTATION

Notation used in this manual:

Numbers with a prefix of ":" (e.g., :7F) are hexadecimal. In 6800/6809 Assembly code, this hexadecimal prefix is shown as "$", consistent with assembler conventions. Numbers without a ":" prefix are decimal.

Bit numbers correspond to the appropriate power of 2; i.e., bit 0 corresponds to :01 and bit 7 corresponds to :80.

Angle brackets around a name denote a class of possible inputs. For example, <filename> means any valid filename.

The notation <CR> is an exception and denotes depressing of the Carriage-Return key on a console device.

Curly brackets in examples indicate optional parameters or phrases to commands; i.e., FILES {TO <device>} indicates that the phrase "TO <device>", is optional.

The notation <class>,... means as many <class> items separated by commas as desired.

A "^" followed by a letter (e.g., ^A) refers to a "control-character", which represents unprintable Ascii character codes in the range :00 to :1F. The actual Ascii code can be determined by subtracting :40 from the Ascii code of the letter following the caret, i.e., ^A represents the Ascii code :01.

Sample typeins are generally in upper case to distinguish them from expository text.

SDOS CONCEPTS

This section contains a short summary of the concepts needed to understand SDOS.

SDOS stands for the Software Dynamics Operating System. The operating system is a (set of) computer program(s) which makes the raw computer hardware much easier to deal with, both for people and for other computer programs.

The term "operating system" actually means two things: in a broad sense, it means the entire set of programs needed to operate a computer, not counting the application programs. This includes a program that is nearly always resident in the computer that lets other programs conveniently converse with peripherals and use the hardware efficiently; it includes a set of utility programs to help the operator of a computer manage the contents of disks and transfer data between peripherals, and it includes program development tools such as compilers, editors, etc. In a narrower sense (the SDOS sense), the operating system means the memory resident program and the utility programs. Sometimes we call the memory resident part the "operating system", because the utility programs generally use it to perform their functions in exactly the same manner as the application programs.

Computer programs generally manipulate data stored on "devices". A device is a (electromechanical) mechanism for storing, acquiring, or outputting data in some fashion; typical examples are disks, video terminals (CRTs), line printers, sensors, etc. Devices are given unique names to distinguish them from one another. Typical device names are DØ:, D1:, D2:, LPT:, CLOCK:, and CONSOLE: (the trailing colon on a device name is an SDOS naming convention for devices).

A "disk" is a rotating magnetic platter used for storing large amounts of data. A "disk drive" is an electronic mechanism for reading/writing data on a disk; a particular disk drive may be used to read or write data on many independent disks at different times. A disk may be removable from a drive so the destinction between individual disks and the drives in which those disks are used is important. The terms "floppy disk" and "disk cartridge" are both represented by "disk" throughout this document.

A "file" is a general concept for a logically related group of data. It may represent a stream of keystrokes arriving from a keyboard of a CRT, data stored in a section of disk memory, or a portion of a magnetic tape. Usually, file refers to data stored on a disk. A disk can generally store many data files.

An "encrypted file" is one whose contents cannot be read or understood without a special key.

A "file name" is an arbitrary name given to a file of data. Usually, the names of the data files are stored on the same

devices as the data itself. To specify a particular data file uniquely, a device name and the file name must be given together. This combination is also referred to as a "file name". Typical file names are: ABC, D1:MYSTUFF.TXT and LPT:.

An "extension" is a suffix of a file name that gives the operator some idea of the type of contents of a file. Extensions are usually set off from the rest of the file name by a special character such as ".". Typical extensions might be .BAS for BASIC program sources, .TXT for raw textual data, .DO for files containing canned sequences of commands, etc. Since file names are arbitrary, extensions are simply a convention; a particular expension does not guarantee the file contains data related to the extension.

A "directory" is a data file used to keep track of file names and the location of data file contents on a device. It acts like a table of contents for files on a disk.

A "bit" is the smallest unit of computer information storage possible and can only represent the values "off" or "on" (interpreted as Ø or 1 respectively). A "byte" is a unit of storage comprised of 8 bits, and can store the code for a single printable character or a number in the range Ø to 255.

A "sector" is the minimum amount of data a disk will read or write and is usually some power-of-2 number of bytes such as 128 or 256 bytes.

A "cluster" is the unit of allocation of disk space to files (the minimum amount of disk space that SDOS will allocate to a file). The size of a cluster is measured in sectors and may be from 1 to 255 sectors.

A "program" is a set of instructions that direct a computer to carry out some operation (computing, printing, sorting, etc.). A "utility" program is one which serves some common need of the operator of the computer, such as a program to list data on a printer, erase unwanted data files, etc. A "command interpreter" is a program which executes a utility function or causes an application program to be executed as a result of operator input.

A "driver" is a special computer program that allows an application program to transfer data to and from a particular I/O device, and to control that device, without requiring the application program to know a lot of detail about how to operate the device mechanics or electronics.

"Protection" refers to a mechanism to ensure the integrity of files or data by preventing the accidental misuse of some action. A "protection bit" is a mark on a file that prevents certain operations from being applied to the file. Examples include protection against writing to a file, etc.

SDOS SYSTEM ARCHITECTURE

This section gives some general details on the structure of SDOS.

First we describe the philosophy of the file system and how it is organized; then we discuss the set of programs which comprise "SDOS"; finally we talk about the structure of the memory-resident portion of SDOS.

Files are a mechanism for storing and retrieving data. SDOS defines a file as a set of data bytes with the first byte being numbered Ø, the second being numbered 1, etc. Data is moved to and from files in variable-length blocks of bytes. SDOS allows two methods of file access: sequential and random. Sequential access allows blocks of data to be read/written to/from successive bytes in the file. Random access allows a file to be positioned to a particular byte so that sequential I/O may start from that point. In effect, SDOS makes a file appear as a huge virtual memory. This technique allows both sequential and random access devices to be treated as similarly as possible, thus increasing device independence.

The contents of a disk can be treated as simply a random access file, or as a set of named disk files, witn each named disk file having the set of properties described above. SDOS keeps track of disk file sizes down to the byte, so that what a program puts into a disk file is precisely what it gets back, no more and no less.

Disk files can be extended dynamically as needed; SDOS will allocate disk space as needed. No explicit guarantee is made that a file occupies a contiguous section of a disk; however, SDOS attempts to allocate disk space in a fashion which "maximizes" the contiguity of a file.

Disk files have names, protection status, and location on a particular disk. No disk file may reside partly on one disk and partly on another. Each disk has its own DIRECTORY.SYS file, which records the names, location, size and other data about all the files on that disk.

SDOS normally handles two kinds of disks: "system" disks and "data" disks. Data disks are used primarily to store data. System disks are required to "boot" (start) and operate SDOS; there is almost always a system disk in some drive on the computer system. Single-drive systems require that all data must live on a system disk.

All disks (system or data) always have the following files present:

DIRECTORY.SYS
BOOT.SYS
DISKMAP.SYS
BADCLUSTERS.SYS

DIRECTORY.SYS is a file which contains the names and some descriptive information about all files on the disk (including itself). This is used by SDOS to translate a file name into the initial information required to locate the data stored in a file.

BOOT.SYS is a file that contains a disk identification, disk tuning parameters such as cluster size, the map algorithm (latency tuning), etc., and on system disks, a boot program that reads SDOS.SYS into memory.

DISKMAP.SYS is a file that keeps track of which parts of the disk are busy or free. It contains one bit per cluster on the disk. A "zero" bit indicates that the corresponding cluster is available for use in creating or extending a file. A "one" bit says that the corresponding cluster is already allocated to a file. If DISKMAP.SYS is not present on a disk, no files may be created, extended, or deleted.

BADCLUSTERS.SYS is the file to which any clusters that contain unreadable or unwriteable (i.e., "bad") data sectors are allocated. Bad clusters are marked in DISKMAP.SYS as allocated so that they will not be re-allocated to other files.

System disks always contain, in addition to the above, the following files:

SDOS.SYS
SERIALNUMBER.SYS
ERRORMSGS.SYS
DEFAULTPROGRAM

SDOS.SYS contains the memory-resident part of the SDOS operating system in SDOS load record format. This file's contents are loaded into memory by the boot procedure; thereafter, the file is not used.

SERIALNUMBER.SYS is a program that contains the serial number and identification of the purchaser of the SDOS software. It is essentially the license for a user to operate SDOS. It is checked once at boot time, and is not used thereafter.

ERRORMSGS.SYS contains the text equivalent of many error codes, and is used to translate the error codes into the text form for display to the operator (this file need not be present for SDOS to run).

DEFAULTPROGRAM is the (user) program that is automatically executed by SDOS whenever any other user program finishes operation or is "killed" by the operator. Normally, it contains a copy of SDOSCOMMANDS, an operator interface package; for turn-key systems, it may contain an application program.

The programs that comprise SDOS consist of the following:

SERIALNUMBER.SYS
SDOS.SYS
SDOSCOMMANDS (DEFAULTPROGRAM)
SDOSDISKINIT
SDOSDISKVALIDATE, SDOSDISKVAL.PAS2, SDOSDISKVAL.PAS3, SDOSDISKVAL.PAS4, SDOSDISKVAL.PAS5
SDOSDISKBACKUP
SDOSERRORMAINT
SDOSSET

SDOSCOMMANDS is a user program which acts as an operator interface. It allows the operator to determine what files are on a disk, to rename, delete, copy or list these files, and to perform miscellaneous other functions. It converts operator commands into sequences of SYSCALLs (see below) which perform these operator's requests. SDOSCOMMANDS recognizes and performs certain commands by itself. All other requests to SDOSCOMMANDS are assumed to be requests to run a program specified in the DIRECTORY.SYS. Usually, a copy of SDOSCOMMANDS has been placed in the file DEFAULTPROGRAM.

SDOSDISKINIT is a user program that takes a freshly formatted disk and sets it up so SDOS can write files on it. In particular, SDOSDISKINIT constructs the files DIRECTORY.SYS, BOOT.SYS, DISKMAP.SYS, and BADCLUSTERS.SYS, on the disk; a vestigial SDOS.SYS file is included in case this disk will be used as a system disk.

SDOSDISKVALIDATE is a user program that verifies and fixes the file structure on a disk; it cannot check to make sure the data is correct. If file structure errors are found, they are reported and the operator is given a choice on methods of fixing the problem. In most cases, the fix results in losing some data; not fixing usually leads to larger data losses at a later time because of a forthcoming disaster. SDOSDISKVAL.PAS2, SDOSDISKVAL.PAS3, SDOSDISKVAL.PAS4 and SDOSDISKVAL.PAS5 are parts of SDOSDISKVALIDATE.

SDOSDISKBACKUP is a user program to make backup copies of entire disks, or subsets of the files on those disks.

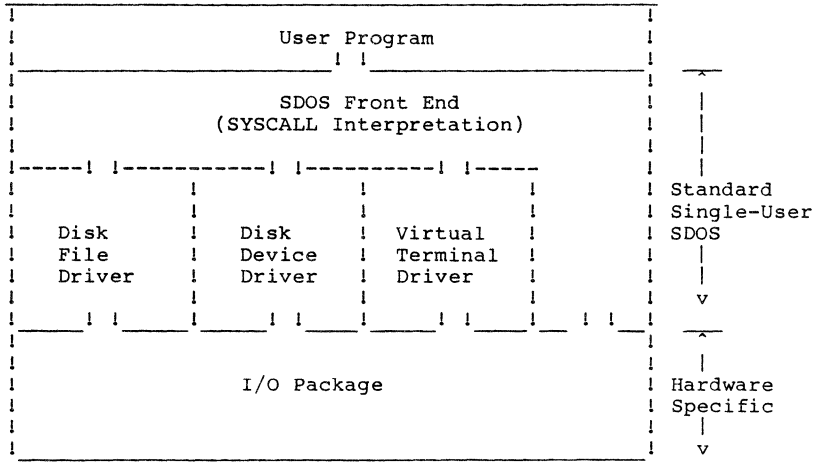
SDOSERRORMAINT is a user program to help the operator maintain the ERRORMSGS.SYS file.

SDOSSET is a user program used to specify the characteristics of a CRT, hardcopy terminal or printer to the Virtual Terminal Driver, eliminating the need to describe such characteristics to each and every application program.

Programs perform I/O and other utility operations via System Calls (SYSCALLs). Each SYSCALL is a subroutine call to the memory resident part of SDOS with a set of parameter data that describes the function to be performed and the data on which the function is to operate.

All of the functions described in the section under SYSCALLs are implemented by the memory resident portion of SDOS.SYS. The memory resident portion is split into several major parts: SDOS Front End, SDOS Disk File Driver, SDOS Disk Device Driver, SDOS Virtual Terminal Driver, SDOS Network Module, SDOS Multi-Terminal Module, and the I/O package. The Network Modules and the Multi-Terminal Modules are options and are discussed further in other sections.

The first four memory-resident parts together are actually single-user SDOS. The other components are used to construct more advanced versions of SDOS, such as multi-user SDOS, or networked SDOS. In many circumstances, we are sloppy and refer to the memory resident part as "SDOS", or even to the entire implementation (utility programs, philosophy, and memory resident part) as "SDOS". The use should be obvious from context.



The SDOS Front End intercepts SYSCALLs, does some initial processing, and then acts as a giant switch, sending the SYSCALLs to the appropriate device drivers. The Front End also contains all the mechanisms that handle interrupts, tasks, etc.

The Disk Device Driver and Disk File Driver are actually integrated with the Front End in the file SDOS11xnnK.68n, and implement the file management part of SDOS. The Virtual Terminal Driver exists in the file SDVT11xnnK.68m, and must be combined with SDO11xnnK.68m and the I/O package software to form a fully functional SDOS.

The I/O package implements all of the non-standard device drivers (CLOCK:, DTOA:, etc.); it contains logical sector I/O routines for the Disk Device and File Drivers; it contains low level routines for performing physical terminal I/O, and it insulates SDOS from all of the particular local hardware peculiarities. The I/O package is designed explicitly to be the place that all user customizing of SDOS is to be performed, and nowhere else; the user may not modify SDOS proper. Alteration of the I/O package requires considerable sophistication on the part of the user.

SDOS COMPONENTS:

SYSTEM FILES:

BOOT.SYS	Disk tuning constants and bootstrap program
SERIALNUMBER.SYS	Holds user's license to run SDOS
SDOS.SYS	Memory resident portion of SDOS
DISKMAP.SYS	Map of allocated clusters (1 bit per cluster)
DIRECTORY.SYS	List of files, file locations, protections, etc.
BADCLUSTERS.SYS	A file which contains only unusable clusters
ERRORMSGS.SYS	Error number to text message conversion

SYSTEM UTILITIES:

SDOSCOMMANDS	SDOS command interpreter with many simple but useful utilities (usually hidden in DEFAULTPROGRAM)
SDOSDISKINIT	Places an SDOS compatible file structure on an empty disk
SDOSDISKVALIDATE SDOSDISKVAL.PAS2 SDOSDISKVAL.PAS3 SDOSDISKVAL.PAS4 SDOSDISKVAL.PAS5	Validates and repairs SDOS file structure
SDOSDISKBACKUP	Makes backup copies of disks or files
SDOSERRORMAINT	Used to examine and modify ERRORMSGS.SYS
SDOSSET	Program to define terminal characteristics to Virtual Terminal Drivers.

DATA FILES:

SDOSSYSGEN*	Installs SDOSBOOT.BIN into BOOT.SYS, SDOSxx.68x plus I/O package into SDOS.SYS
SDOSUSERDEFS.ASM	To be added to any user-written assembly code requiring Syscalls
SDOSIOPKDEFS.ASM*	For use with I/O package generation
SDOSIOPACK.ASM*	I/O package source
MAKEVTCFG*	Program that automatically manufactures part of the I/O package for the Virtual Terminal driver.
SDOS11xnnK.68m*	SDOS object
SDVT11xnnK.68m*	Virtual Terminal Driver object
IOVTDPBS.ASM*	Virtual Terminal Device profiles, source form
SDOSBOOT.ASM*	Source for Bootstrap program
SDOSBOOT.BIN*	Object for Bootstrap program
ERRORMSGBUILD.DO	DO file that builds ERRORMSGS.SYS
SDOSCMDS.BAS*	SDOSCOMMANDS source: BASIC part
SDOSCMDS.ASM*	SDOSCOMMANDS source: ASSEMBLY part
SDOSCMDSGEN.DO*	DO file to build SDOSCMDS.BIN
SERIALIZE*	A program to encrypt an application program to ensure operation on only a single computer.

REQUIRED PROGRAMS:

BASICRTPV14.BIN	BASIC Compiler Runtime Package

* Optional SDOS customizing package. Names may vary.

SYSTEM OPERATION OVERVIEW

An SDOS session consists of booting SDOS, running applications or utilities, and finally, shutting the system down (important: see SHUTTING DOWN).

Booting is used to bring a copy of SDOS from a disk into the memory of the computer where it stays for the duration of the session.

Once SDOS is in memory, it loads the DEFAULTPROGRAM and runs it as an application program (SDOS does not run programs with any special privileges or any special modes of operation). On a general purpose or development system, the DEFAULTPROGRAM contains an operator command interpreter, which allows the operator to perform various utility operations and cause the execution of an application program or development tool (such as a compiler). On turn-key systems, DEFAULTPROGRAM contains an application program (generally a menu-driven sub-application selector). The DEFAULTPROGRAM may cause another application or utility program to be loaded and executed (perhaps by operator command). When an application/utility program is done, it does an (ERROR) EXIT to SDOS, which re-loads the DEFAULTPROGRAM and so starts the cycle again.

Conversations between a program and the operator are (by convention) done via I/O channel number zero (which is normally OPEN to the CONSOLE: device; SDOS opens channel zero to the CONSOLE: whenever it finds a read/write request to channel zero with channel zero closed). EXIT closes all I/O channels except zero. This allows whatever file has been opened for operator input to be passed from one program to another, and is the basis for DO files.

Errors which occur during execution of an application program are reported by SDOS to that program via an error code. The program may process and recover from the error itself, or it may pass the error code back to SDOS for display (via an ERROREXIT).

BOOTING SDOS

"Booting" refers to the process of starting computer operations under an operating system. For application systems, this process is usually done once a day. For development systems, booting may be more frequent.

To get SDOS started, the operator needs to perform the following steps:

1) Ensure that power is on to the computer system, disk drives and the operator's console. Some systems have other peripheral devices that need to be powered up in order for SDOS to boot properly.

2) Insert a "system" disk into the disk drive which will be used as the system boot device. Note: This disk must have valid BOOT.SYS, SDOS.SYS, DIRECTORY.SYS, SERIALNUMBER.SYS and DEFAULTPROGRAM files on it, or the boot process will not succeed! Disks with the needed files are generated properly by the SDOSDISKINIT program.

3) Push the RESET switch on the computer. ALWAYS push RESET before booting; this puts the computer in a known safe state. Depending on your system configuration, one of three things can happen:

   A) If your system has Software Dynamics IDB in ROM, the message

      IDB Vx.y

      will appear. The operator must type "G" to continue the boot process.

   B) If your system has no "monitor" program of its own, the boot ROM in the computer will take over automatically and read in SDOS from your disk.

   C) Some systems have manufacturer-specific monitor programs. The boot procedure for these systems is monitor dependent, but usually consists of some form of computer memory address entry followed by a "GO" command of some kind. See the manufacturer's documentation.

Some systems, with more than one kind of disk drive (i.e., a mixture of floppy and hard disks) may ask the operator which drive to boot from. Again, see the manufacturer's documentation.

At this point, the boot process should have taken over automatically. There will be a short burst of activity on the chosen (system) disk drive, and then the following banner message will appear:

SDOS, Version 1.1g Copyright (C) 1978 Software Dynamics

This message signifies that SDOS has managed to successfully load itself into memory, and has started operations.

Immediately thereafter, a message of the form:

mm/dd/yy ...text...

will appear. This is the date that the system disk was generated; the text is the disk identification that was given to SDOSDISKINIT when the disk was initialized.

If this message contains the word "MASTER" anywhere, you should not use the disk for normal operations; only for initializing another disk and/or recovering from disasters. It is better to preserve a MASTER disk (obtained from the vendor) in a safe place, and use a backup copy in case something goes wrong. Backup disks can be made with the SDOSDISKBACKUP program.

Next, the message

This copy licensed for use only by PURCHASERNAME
only on the single computer with CPU Serial Number xxxxxxxxxxxxxxx

will appear. This message shows who is licensed to use this copy of the software. If the PURCHASERNAME does not match that of the organization, then the copy is probably illegal and should be reported to Software Dynamics.

Finally, a "." prompt should appear (if DEFAULTPROGRAM contains SDOSCOMMANDS). The dot is printed out by the operator interface program, SDOSCOMMANDS. Immediately following the dot, SDOS will prompt the operator for the time of day (see TIME command under SDOSCOMMANDS), if the computer hardware does not remember. Entering the time and date completes the boot process, and normal use of SDOS may now start.

Sometimes, much displayed output will occur before keyboard entry is allowed; this happens when a file, INITIALIZE.SYS, has been set up to tell the system what to do every time it boots.

In a turnkey system, the prompt displayed is application program dependent.

Several things can go wrong during the booting process. In step 3, no reaction at all might occur in response to RESET. This means your computer is probably sick, not powered up, etc. During the automatic part of the boot, dead silence may ensue.

There are several possible causes: the desired disk drive is not powered up, not ready, or does not have the disk seated in it properly; or you may have told the computer to boot from the wrong drive (operator errors).

A (software) damaged or improperly generated system disk, or use of a disk that only contains data files as a system disk, will also cause dead silence, as the required programs to complete the booting process are not present, and the computer cannot do anything without those programs. If this appears to be the case, try booting a backup of the MASTER. If the backup of the MASTER will not boot either, then you probably have a problem in your CPU, its memory, or the disk drive. If the MASTER backup boots, then the original disk you tried to boot from is probably software damaged, etc. The SDOSDISKVALIDATE program may be able to repair a software damaged disk.

If you get the SDOS banner, but no disk identification, your system has a serious problem, because the same routine that read in SDOS was able to do so only by first reading the disk sector containing the disk identification.

If no "This copy licensed..." message appears, the boot disk is missing or has a bad copy of SERIALNUMBER.SYS. Attempting to boot a disk intended for another computer will get "Can't run on this serial number" and operation of SDOS will cease.

If the banner, disk identification and serial number appear, but no "." or prompt appears, DEFAULTPROGRAM on this disk is probably damaged.

Error 1Ø45 (disk read), or error 1Ø47 (disk seek) appearing during the boot process means your disk is probably worn or software damaged.

Error 1Ø08 means DEFAULTPROGRAM cannot be found on the disk.

If you have any of these problems, it is a good idea to push RESET quickly after the problem is discovered to minimize any further software damage caused by the malfunction.

Any other error messages that occur indicate a software malfunction and should be reported as a possible bug.

USING THE KEYBOARD ON THE OPERATOR'S CONSOLE

This section generally describes the various keystrokes that have special meaning to SDOS. Uniform interpretation of these keystrokes, across a wide variety of terminals, is ensured by Device Profile Blocks, in the I/O package. For more details, refer to the section on the Virtual Terminal Driver in the Application Programmer's Guide.

Input Line Editing:

Virtually all commands and data entered via the keyboard into SDOS or a program operating under SDOS are done in "line mode". This allows the typist to enter the complete command/datum, to correct the input, as required, and review the input data for correctness, before the entire input line is handed over to SDOS or the program running. The typist indicates his satisfaction with the entered data by depressing the carriage-return (RETURN or <CR>) key on the keyboard. Prior to doing this, he may correct the entered line using control characters described below. Once the RETURN key is pressed, there is no way to prevent the entered line from being given to SDOS or the program. Once input is requested, no action is taken by the program until <CR> is depressed. On CRTs, control characters allow the typist to move the cursor about WITHIN the entered data in order to correct errors.

Control characters used to edit input lines:

^E	ERASE; erase all input at, and to the right of, the cursor
^F	FRONT; moves cursor to beginning of line if CRT device
^H	BACKSPACE; moves the cursor backward; beeps if the cursor is at the beginning of the line
^I	TABS the input; passed to the program as a tab character
^L	FORESPACE; moves the cursor forward; beeps if the cursor is at the end of the line
^M	CARRIAGE RETURN; causes the entire input line to be passed to the program
^R	RETYPES the part of the line entered so far if this is a hardcopy device; RIGHT; moves cursor to end of line if CRT device
^U	DELETES the character which is under the cursor, erases it from the display and shifts all characters to the right of the cursor one position to the left; beeps if the cursor is at the end of the line
^X	CANCELS the line entered so far; the typist must completely re-enter the line
RUBOUT	DELETES the character to the left of the cursor, erases it from the display and shifts all characters to the right of the cursor one position to the left; beeps if the cursor is at the beginning of the line

Field Editing:

When performing entry for a display-oriented application which uses fields (a specific place on the screen, for data entry), editing is slightly different than with input line editing. The differences are determined primarily by the application, but are generally as follow:

^H	(Left Arrow) moves the cursor left within the field. At the left end of the field causes field data to be passed to the application, along with a "GO LEFT TO NEXT FIELD" indication.
^J	(Down Arrow) causes field data to be passed to the application, along with a "GO DOWN TO NEXT FIELD" indication
^K	(Up Arrow) causes field data to be passed to the application, along with a "GO UP TO NEXT FIELD" indication
^L	(Right Arrow) moves the cursor forward within the field. At the right hand end of the data causes field data to be passed to the application, along with a "GO RIGHT TO NEXT FIELD" indication.

The field has an upper limit on its width. Typing into the last character position of the field (filling the field) may either be illegal (unless an activator character, such as <CR>, is entered), or may cause the field entry to be terminated, and the data to be passed to the program. Refer to the description of the particular application program, for more detail.

Special Control Characters:

Special control characters are used to interact with SDOS or the BASIC Run-Time Package to perform various functions. None of these special characters are passed to a program requesting input.

^A Toggles "fold" mode. In fold mode, lowercase letter keys are translated automatically to uppercase. When not in fold mode, lowercase letter keys are passed to programs as lowercase.

^B Used to set breakpoints on line numbers in BASIC programs. See BASIC Manual.

^C A single ^C clears typeahead buffer and removes any output freeze; use it if a mistake is made during typeahead which cannot be corrected by RUBOUT, or if no output is coming out a terminal when it is expectd. ^C^C will cause any program to be killed. A ^C will not be accepted if the program is KILLPROOF (instead, a beep will sound). Logging is terminated, and any DO file is aborted. Two ^C's must be typed in succession (to prevent the typist from accidentally killing a program). When the first ^C is typed, the type-ahead buffer is cleared, the ^O, ^P and ^S modes are exited, and ^C is echoed immediately. The second ^C will not kill the program if any other key was struck since the first ^C. This allows the typist to determine that SDOS has not completely died, by typing ^C<LF>; the ^C will echo, and the <LF> will prevent the next ^C from killing the program running. No ^C echo (and no beep) is a good sign that SDOS has crashed.

^D Causes the system debugger to get control, as though a non-maskable interrupt had occurred. If running under /MT, the currently running application is KILLPROOF or encrypted or no debugger is present, ^D echoes a beep.

^G Used to go from a breakpoint in a BASIC program. Also exits ^V mode.

^O Used to discard output; output is discarded until another ^O, a ^Q, or a ^C is typed, or input is required.

^P Toggle page mode display. If the page mode toggle is on, SDOS will print the next screenful of output lines on the display, print ^P at the bottom right of the screen, and then stop output. The typist may then type a ^Q to see the next screenful of lines, or ^P to leave the page mode (which will cause SDOS to print without pausing for typist intervention). The application program is frozen until ^P or ^Q is typed. Page mode is normally used when listing a large file on the terminal, and the typist wishes to inspect the listing closely.

^Q Continue output (see ^O, ^P, ^S).

^S Stop output now. Used by the typist to temporarily stop the computer from printing more text on the terminal. ^S will be printed at the bottom right of the display, and the typist must type ^Q to allow the output to continue.

^T Trace the line numbers of a BASIC program.

^V Single step the lines of a BASIC program.

^W Causes last input line to be retrieved as though the typist had entered it explicitly, if no other keys have been typed since the last input. Illegal for hardcopy devices.

^Z Causes an end of file condition to occur on the terminal, if typed in response to an ASCII read request.

<ESC> Signifies that the typist would like to interact with the program. This allows the typist to signal a busy or compute-bound program an attention request, without killing the program. The program can sense a typist attention request, and process it at its leisure. There is no guarantee that a particular program pays any attention to a typist attention request.

Type-Ahead

All keys (except the special control characters) struck by the typist, when the currently running program is not waiting for input, are not echoed, but are saved in a "type-ahead" buffer. The characters are processed and displayed when input is required, as though the typist had entered them then, and not earlier. This allows the typist to get ahead of the program's input requests, if he knows what data will be needed next. A single ^C will cancel all the type-ahead.

Binary Input Mode

Some programs operate in "binary" input mode. In this mode, all keystrokes, including the special control characters, are given to the program as-is. ^C^C will not kill a program that uses this mode. No input editing is possible, without the program's aid, so editing is thus program-dependent. The majority of programs operate in line input mode, as described above (not in binary mode); unless otherwise noted, all programs operate in the line input mode.

DEVICE and DISK FILE NAMES

Stored data is given a name (by the user) so that he may later retrieve that data. This name is known as a "file" name. Data may also be read or written to a peripheral device; to indicate which device, a "device name" is used. A single device (such as a disk) may be able to store many files; in this case, the device name and the file name must be given together to select the proper file. The combination of device name and file name is also called a "filename".

A device name is composed of any sequence of alphanumeric characters followed by a colon; the first character must be alphabetic. Lower case alphabetic characters are treated as being equivalent to their upper case version. The device name is generally a mnemonic related to the actual English name of the device, with an optional trailing digit if more than one of that kind of device may be connected to a system. Disk device names are short because they tend to be typed frequently. A misspelled device name will be promptly caught by SDOS.

Typical device names are:

CONSOLE:	The user's console. Available in all SDOS systems.
DØ:, D1:	Disk Ø, 1, 2 ... One name for each disk unit.
DISK:	Name of default disk (see DEFAULTDISK command of the command interpreter).
LPT:	Line Printer.
CLOCK:	The time and date device.

Disk file names have the following form:

```
        filename
or
        filename(integer)
```

The filename must be from one to sixteen characters, from the set $, ., A-Z, Ø-9, or a-z (lowercase is automatically treated as uppercase). The first character of the filename must be $ or A-Z (not a digit or "."!). The optional "integer" in parentheses is used at file creation time to allocate enough disk space to contain the number of data bytes specified by "integer". Names longer than 16 characters (excluding the parenthesized file size) are not legal and will be rejected.

Typical disk file names:

```
MYFILE
PAYROLL.BAS
MONTHREPORT.LPT
D3:ABC
DISK:EDIT
```

FILENAME EXTENSIONS

An "extension" is an agreed-upon suffix to a file name that gives some useful information about the contents of that file. SDOS disk filename extensions consist of a period followed by one or more letters, limited only by the size of filenames. A particular extension indicates a particular file type. An example might be ABC.TXT; "ABC" is the name by which the user would like to refer to the file; .TXT tells him that the file contains raw text (as opposed to, say, a computer program or list of prime numbers). Since an extension is merely part of the filename, and files can be named (or renamed) arbitrarily, these extensions are merely conventions. Their utility is directly proportional to the amount of energy invested by the user in sticking to the conventions.

The following extensions are defined and used by standard SD products:

.EXE	For executable program binaries.
.DO	For command ("DO") files
.MIK	For files containing MIKBUG object records
.BAS	For files containing the source of BASIC programs
.BAK	For slightly older revisions of a source file
.TYP	For files containing data intended to be processed by the optional TYPE program
.TMP	For temporary files
.68Ø	For object files intended for execution on 68Ø0 only
.689	For object files intended for execution on 68Ø9 only
.DOC	For files containing text to be fed to a document formatting program
.ASM	For files containing assembly source code
.TXT	For files containing raw text
.DAT	For files containing data other than text
.LPT	For files containing listings meant for a printer (i.e., an LPT:)
.BIN	For files containing SDOS load records but that are not intended for independent execution.
.SYS	For files containing SDOS system data
no extension	For executable programs, i.e., for SDOS load record binaries
.CM	For executable object files intended for operation under MDOS (TM Motorola)

PROTECTION BITS

A protection bit is used to prevent certain operations from being applied accidentally or maliciously to a file. SDOS provides two kinds of protection: Write Protection and Backup Protection. Write protection prevents a file from being altered, deleted, or renamed. Backup protection prevents a file from being backed up redundantly.

THE COMMAND INTERPRETER

The command interpreter (SDOSCOMMANDS) is a utility program which provides many simple but frequently used file manipulation commands. It also allows the operator to cause the execution of any program, to perform canned sequences of operator commands, and to perform some miscellaneous utility operations. The command interpreter also provides a simple, relatively consistent format for passing parameters (such as file names) to user programs.

On most SDOS systems, a copy of the command interpreter is usually stored in the DEFAULTPROGRAM file, so that whenever an application program finishes execution, the command interpreter is loaded and begins execution. Turn-key systems built around SDOS usually have a particular application program stored in DEFAULTPROGRAM with some method to allow the operator to execute SDOSCOMMANDS (which contains the command interpreter).

The command interpreter prints a "." to indicate it is ready to execute another command. The examples show this dot, but it is not typed in by the operator. The command format expected is:

```
        <command> <parameters>
or
        <command>
```

where the parameters are separated from the command by one or more blanks. Some commands require no parameters; in this case, information in the parameter field is ignored. Other commands will select a default set of parameters if an empty parameter field is given.

Some commands (and many programs) which require parameters can be invoked by merely typing their name. Such commands will discover that no parameters were given, and will prompt the operator for the needed input (this is known as VERBOSE command mode; when parameters are given on the same line, it is called TERSE command mode). When in doubt, type just the command name; it will prompt if more input is needed.

Input to the command interpreter is done in line mode. The operator must push the <CR> key to cause the command interpreter to act. All editing keystrokes are valid. The command interpreter treats all type-in as though it were typed in upper case; the examples are shown in upper case.

Commands handled by the command interpreter are:

FILES	List the names of files on a disk
LIST	List the contents of a particular file
COPY	Copy the contents of a file from one place to another
RENAME	Change the name of a file
DELETE	Make a file disappear
DISMOUNT	Causes SDOS to "let go" of a disk
MOUNT	Notifies SDOS of presence of a disk
DEFAULTDISK	Directs SDOS's attention to a particular disk
TIME	Set/display time and date
FREE	Display amount of available space on a disk
SETPROTECTION	Enables protection of a file against certain operations
CLEARPROTECTION	Allows operations on a file formerly prohibited by SETPROTECTION
VERSION	Displays version number of command interpreter
HELP	Converts an error number to a corresponding text message
DEBUG	Load a test program and give control to debugger
CRC	Used to get a "signature" number that is dependent on file content, order, and length.
USERSPACE	Used to determine amount of program space in computer.
LOG	Makes copy of console session and places in a file
CLOSELOG	Stops copying console session
DO	Execute a canned sequence of commands
LABEL	Target point of GOTO or IFERROR
GOTO	Skips over canned commands
IFERROR	Conditionally skips over canned commands
*	Comment line

If a command is not recognized, it is assumed to be the name of a program (file) to be executed. Most of the complex utility programs (such as SDOSDISKINIT and SDOSDISKVALIDATE), along with user programs, are invoked in this fashion, thus allowing invocation of "commands" external to SDOSCOMMANDS and commands internal to SDOSCOMMANDS in the same fashion. Parameters given to commands not recognized by SDOSCOMMANDS are passed to the program specified as the first line of console input (i.e., the first READA (or INPUT) of a program will read the part of the input line not occupied by the command name) (see CC:GETCOL).

FILES

The FILES command is used to determine what disk files reside on a disk. It will also display a subset of the files whose names match a pattern given by the operator.

The FILES command has the syntax:

```
        FILES
or
        FILES <device>
or
        FILES <device> <filename pattern>
```

<device> is intended to be the name of a disk drive (such as DØ:, D1:, etc.). If <device> is not given, DISK: (the default disk; see DEFAULTDISK command) is assumed.

The <filename pattern> is used to select which filenames on the specified disk are to be displayed. The <filename pattern> consists of any valid filename, with portions replaced by an "*". The "*" is known as a "wildcard", and is used to signify any sequence of zero or more file name characters. Filenames will not be displayed by the FILES command unless they match the pattern given. A match occurs when a filename under consideration has all of the (legal) characters given by the pattern, in the order specified by the pattern. Wildcards are used to match the rest of the filename. Wildcards may occur at the beginning, end, or middle of a pattern; multiple wildcards are allowed. Doubled wildcards (**) are treated as single wildcards. Thus, A* matches ABC, APE.BAS; B*.ASM matches BOUND.ASM and B.ASM; *EN* matches any filename that contains the letters E and N adjacent; *E*N* matches any filename that contains an E followed eventually by an N. If no <filename pattern> is given, all filenames of files on the selected drive are displayed (i.e., * is used as the <filename pattern>).

The FILES command displays the identification of the disk specified, one line per filename, and the percentage of the disk space occupied by the files displayed.

Each filename displayed is displayed with data concerning the physical disk space occupied (LCNs), the virtual disk space occupied (BYTES), file protection codes, and the date of creation of the file.

LCNs give the space allocated to a file in terms of clusters.

BYTES describes the highest number data byte written to the file. Note that LCNs is not necessarily a direct function of Bytes due to the possibility of a file being sparse (see SDOS DISK FILE STRUCTURE and also, ERRORMSGS.SYS in example below).

The protection codes are listed as D for a delete protected file; W for write protected, B for backup protected, and blank for no protection.

Sample directory listing:

```
.FILES D1:*S*
Files on...  FRED'S GAMES DISK
Filename        # LCNs     Bytes   Prot     Date
SDOS.SYS            28     23851     W    Ø8/Ø3/83
BASIC               12     11147     W    Ø8/Ø3/83
STARWARS             7      5283          Ø9/12/83
GALAXIES             2       611          Ø9/27/83
STARTREK             5      3554          1Ø/Ø2/83
STRATEGY             6      4949          1Ø/22/83
ERRORMSGS.SYS       1Ø    199655          11/Ø7/83
Total of 7Ø clusters in 7 files for 22.7% of disk capacity
```

Hitting ESCape during a FILES listing will abort the command.

The form:

FILES <pattern> TO <filename>

allows a FILES listing to be placed on a printer, or into a file, as specified by the filename following the word "to". The word "to" must be separated from the <pattern> and the <filename> by at least one blank on each side.

Note that the FILES command may not display the correct current values of the file attributes if the file is currently in use by some other program (this can only occur in multi-user or network systems).

LIST

LIST is used to quickly scan the contents of a file containing ASCII text, or to copy a text file to some printing device to obtain a hard copy.

The LIST command has the format:

```
        LIST <filename>
or
        LIST <filename1> TO <filename2>
```

The word "TO" must be separated from the <filename>s by at least one blank on each side.

The first form is treated as though LIST <filename> TO CONSOLE: had been typed instead; this prints a copy of the file on the operator's CONSOLE:. The file may be inspected at whatever rate is appropriate for the operator by judicious use of the ^P, ^S, and ^Q keys (for paging control).

The second form causes the command interpreter to CREATE the file specified by <filename2>, and copy <filename1>'s contents to <filename2> using ASCII line-mode reads and writes. This is particularly convenient when used as follows:

```
        LIST <filename1> TO LPT:
```

which causes the selected file to be printed on the line printer.

When building a small text file (this is especially handy for DO files),

```
        LIST CONSOLE: TO WHATEVER.DO
```

is a convenient way for the operator to key in the text for the DO file directly without use of the EDITor. Exit from this mode of data entry is accomplished by typing ^Z, which signifies end-of-file for the CONSOLE: device.

The following will copy a text file from a paper tape reader to a disk file:

```
LIST READER: TO MYFILE.TXT
```

LIST can be used to copy a text disk file to another disk file; but it cannot be used to copy a non-text file, because LIST will "interpret" (expand tabs, insert ASCII:LF after ASCII:CR, etc.) control codes. Since the COPY command will copy either text or non-text disk files, and is generally faster than LIST, moving copies from one disk file to another is generally done only with the COPY command. The LIST command is generally useful only when an I/O device other than a disk is involved as a source or a target.

Hitting ESCape will abort the LIST command.

COPY

The COPY command is used to make exact copies of disk files or the data received from an I/O device. It can also be used to perform a simple disk backup or to append several files together.

The form of the COPY command is:

```
        COPY <sourcefile> TO <destfile>
or
        COPY <source1>,<source2>,... TO <destfile>
```

A new copy of <destfile> is CREATEd (so an old file by that name will be lost; no warning is given), and the source files are copied in the order specified into the newly created file. The first source file is opened before the destination file is created. The copy is performed using binary reads and writes, so that the file contents are copied exactly, byte for byte.

The COPY command uses all available memory as a large buffer to optimize the COPY; this makes COPY move data considerably faster than a very simple, one-byte-at-a-time copy program.

COPY will copy sparse files to another disk file, preserving the sparseness property, but it will not preserve the sparsity if the target file is not a disk file. It will not necessarily preserve the exact structure of the sparseness, so the number of LCNs in the copied file may not match the number in the source exactly. COPY preserves the sparsity by simply positioning past large blocks of zero data bytes in the source file.

If the destination is a disk device only (i.e., not a file on a disk) then COPY will ask for a verification before it proceeds; this prevents accidental copying onto a file-structured disk with the consequent disasterous results of destroying the file structure on the target disk.

```
        .COPY PRIME.BAS TO D1:PRIME.BAS
```

moves the file PRIME.BAS from the default disk to D1: (copying multiple files from one disk to another is more easily accomplished via the SDOSDISKBACKUP program).

```
        .COPY FIRSTPART.ASM,SECONDPART.ASM TO WHOLETHING.ASM
```

appends FIRSTPART and SECONDPART together. No data bytes are inserted between the two parts. This is particularly useful when reconstructing files that have been SPLIT by SDOSDISKBACKUP.

```
.COPY ERRORMSGS.SYS TO D1:ERRORMSGS.SYS
```

copies the sparse file ERRORMSGS.SYS. Note that absolutely no operator action is required to preserve the sparseness.

If an * is used as the filename part of <destfile>, then the filename part of <sourcefile> is used as the destination filename. If <sourcefile> is a list of files, or simply a device name, using * in <destfile> is not legal.

```
.COPY PRIME.BAS TO D1:*
```

copies PRIME.BAS from the default disk to D1:.

To place a file on a disk without any file structure (as though the disk were a paper tape, with LSN Ø being the first block, LSN 1 being the second, etc.) the following needs to be done:

```
.DISMOUNT Dn:
.COPY file TO Dn:
Are you sure you want to write on the disk DEVICE? YES
```

The DISMOUNT command forces the map algorithm on Dn: to become "1" (a convenience when later trying to read the disk).

WARNING: This type of COPY destroys the file structure on the destination disk; note the verification required before COPY will begin. If you don't understand what this means, type NO or you will learn about it very painfully.

Recovery of a file written onto a disk as above is effected as follows:

```
.DISMOUNT Dn:
.COPY Dn: TO AFILE
```

This will recover the file; unfortunately, it will also copy the unused part of Dn: into AFILE so that special editing of AFILE is needed to complete the process. Text files written onto a disk device are usually recovered by using the EDITor to "edit" the text from the disk device.

A simple disk backup scheme is effected as follows:

```
.DISMOUNT Dn:
.DISMOUNT Dm:
.COPY Dn: TO Dm:
Are you sure you want to write on the disk DEVICE? YES
.DISMOUNT Dm:
```

This copies Dn: to Dm: (SDOSDISKBACKUP is the recommended method of backing up a disk). Neither Dn: nor Dm: need to have a valid SDOS file structure; any disk compatible with the drive can be copied in this fashion.

The CRC command (which computes checksums over files and devices) can be used to verify that both disks contain identical values:

```
.CRC Dn:
 CRC = :xxxx
.CRC Dm:
 CRC = :xxxx
```

If the source disk contained a valid SDOS file structure, SDOSDISKVALIDATE can be used on Dm: after the copy is complete to change the disk identification.

A trap many users fall into is:

```
.COPY CONSOLE: TO FILE
```

There is no way out of this but <RESET> on the computer.

RENAME

The RENAME command is used to change the name of a disk file. The file is not moved or modified in any way.

The RENAME command has the following form:

RENAME <oldfile> TO <newfile>

Only disk file names are allowed. Renaming a file to a device name is illegal, as is renaming a file on one disk to a filename with a different disk specification. If Dn: is specified with <oldfile>, it need not be specified with <newfile>.

RENAME ABC.TXT TO PRIMES.BAS

changes the name of the file ABC.TXT on the default disk to PRIMES.BAS (on the default disk).

RENAME D2:TESTDATA TO LIVEDATA

renames TESTDATA, a file that is on D2: instead of on the default disk.

The RENAME command can also be used to change the identification of a disk. The form is:

RENAME <diskdevicename> TO <disk identification text>

This changes what is printed as the disk id by the MOUNT or FILES command. The <disk identification text> must be 32 characters or less.

Example:

RENAME DØ: TO MASTER PAYROLL DATA

DELETE

The DELETE command is used to erase the names and contents of a specified set of disk files. The space used by those files is returned to available space on the disk that contained the file, for re-use later when more files are created or extended.

The form of the DELETE command is:

DELETE <file1>,<file2>,...

The specified list of filenames is examined and each is deleted in turn. A device specification will ensure that the file to be deleted was really on the specified disk. If a specified file cannot be found, or an error occurs, the DELETE command complains and ignores the remainder of the list.

A filename may contain wildcards (see FILES command). The DELETE command will delete all files whose names match the pattern given. The deletion process can either be automatic or verified in each individual case; the latter allows selective deletion. When the DELETE command discovers a wildcard for the first time, it displays:

Ask before doing the delete?

A response of N or NO will cause DELETE to find all filenames that match the pattern, delete the corresponding files, and list the names of the deleted files. Any other response is interpreted as YES; this causes the DELETE command to ask

Delete <filename>?

for each filename found that matches. A response beginning with Y to this question will cause <filename> to be deleted; any other response will cause <filename> to be left intact (i.e., not deleted).

If another filename in the delete list is encountered containing a wildcard, the DELETE command deletes matching files in the verification mode supplied the first time it asked

"Ask before doing the delete?"

The wildcard delete can be an enormous timesaver if used properly. It can be a disaster if used carelessly! Beware.

```
.DELETE D2:ABC
```

does what it says; ABC is deleted on device D2:.

```
.DELETE D1:*.BAS,D2:Q.TMP
Ask before doing the delete? Y
Delete JUNK.BAS? Y
Delete USEFUL.BAS? <CR>
Delete OTHER.BAS? YES
```

This sequence deleted D1:JUNK.BAS, D1:OTHER.BAS, and D2:Q.TMP; USEFUL.BAS was retained.

```
.DELETE *A*
Ask before doing the delete? NO
JAM.TXT
INVENTORYDATA
TRASH.JNK
```

Note that the file INVENTORYDATA was deleted; if this is what the operator intended, fine; if not, he should have been more careful and used the verify option.

DISMOUNT

The DISMOUNT command is used by the operator to notify SDOS that he is about to remove a disk from a disk drive. This occurs when a different disk is desired, or when shutting the system down.

WARNING: FAILURE TO USE THE DISMOUNT COMMAND BEFORE REMOVING A DISK FROM A DRIVE MAY RESULT IN LOST DATA OR A DAMAGED FILE SYSTEM ON THAT DISK!

Replacement of one disk by another without notifying SDOS with the DISMOUNT may damage data on BOTH disks!

The form of the DISMOUNT command is:

```
DISMOUNT <diskdevicename>,...
```

This command causes SDOS to write all modified disk sectors (for the specified disks) that remain in the computer's memory back to the specified disks, thus ensuring its integrity (this command also causes SDOS to "let go" of the system files DIRECTORY.SYS, DISKMAP.SYS, and ERRORMSGS.SYS on the specified disks, and forget about any unmodified disk sectors it may have in memory).

In an effort to prevent system crashes or the operator from accidentally damaging his disks, SDOS does write all data that belongs to a disk back to that disk when an application program stops (EXITs). This means that when the "." is first printed by the command interpreter after execution of any program, the data and file structure of all disks is safe and completely up to date (note: see SDOS/MT documentation).

There is a corresponding MOUNT command that notifies SDOS of the presence of a new disk, but use of it is not generally necessary; SDOS does an implied MOUNT when its attention is directed to a disk drive it thought was dismounted.

```
.DISMOUNT DØ:
```

releases DØ:; the operator may remove the disk in DØ: when the "." prompt is printed after completion of the command (NOT BEFORE!). If a "Write Protect" error occurs during a dismount, the operator should repeat the dismount until the error no longer occurs and then run SDOSDISKVALIDATE on the disk.

Note: Before shutting the system down (at the end of the day or before powering the computer off), all drives containing disks MUST be dismounted.

The DISMOUNT command will display disk I/O error statistics if the disk sector I/O driver makes them available. If no errors have occurred during disk usage, the following display is typical:

```
.DISMOUNT D1:
A total of 64 I/O operations on D1:
```

Dismounting multiple drives gives appropriate multiple messages:

```
.DISMOUNT WDØ:,DØ:
A total of 102763 I/O operations on WDØ:

A total of 722 I/O operations on DØ:
```

If errors have occurred, then a different display results:

```
.DISMOUNT FØ:
Error totals on FØ:

Operation          Error count        Status
---------          -----------        ------
Read                23                :19ØØ
Write               Ø                 :ØØØØ
Seek                Ø                 :ØØØØ

A total of 23 I/O errors out of 287 I/O operations
Last Bad Sector (Soft) = :ØØØØFB
Last Bad Sector (Hard) = :ØØØØEB
```

The actual meaning of the counts and the status is disk driver dependent and so requires special knowledge to interpret. However, the operator can get a "feel" for error counts that are "normal" for a disk; if displayed values are seriously out of line, then the disk cartridge or drive is beginning to have a problem; this should be investigated and fixed before serious damage results. As a general rule, no error should occur during normal operation.

MOUNT

The MOUNT command is used by the operator to notify SDOS that a new disk has been placed into a drive. The form is:

.MOUNT <diskdevicename>

SDOS will do an automatic MOUNT if a drive which has been DISMOUNTed is referenced, so generally this command is not needed. However, since it prints the disk identification, it can be useful to see which disks are actually being mounted.

DEFAULTDISK

The DEFAULTDISK command is used to tell SDOS which disk drive is to be used when a filename without a specific device prefix is given. The form is:

```
DEFAULTDISK <diskdevicename>
```

When SDOS is booted, it selects a DEFAULTDISK which corresponds to the device from which it was booted (this is usually named DØ:). This allows un-prefixed file names to automatically refer to files on the boot device; that is, ABC would really be interpreted as DØ:ABC (D3:DEF is interpreted as file DEF on D3: because of the explicitly given device prefix). The device name DISK: is a dummy name for the currently chosen DEFAULTDISK (i.e., ABC is the same as DØ:ABC is the same as DISK:ABC in this case).

When the operator discovers he is making many references to files on a disk drive other than the boot device, he can minimize his typing (of devicenames) by changing the DEFAULTDISK to the drive he is using frequently. This is done by typing

```
.DEFAULTDISK D2:
```

when D2: is the disk drive which contains the files he is referencing frequently. All further references to ABC will then mean D2:ABC instead of DØ:ABC as it was previously.

The newly chosen default disk must have all the desired programs on it (including DEFAULTPROGRAM), or the operator will have to prefix the program name with the appropriate device. For instance, if a PAYROLL program is stored on DØ:, and the default disk is currently D2:, to run the PAYROLL program, the operator must type

```
.DØ:PAYROLL
```

When changing the default disk device, the following is required to ensure that SDOS looks in ERRORMSGS.SYS on the new default disk:

```
.DISMOUNT <olddefaultdisk>
.DISMOUNT <newdefaultdisk>
.DEFAULTDISK <newdefaultdisk>
.MOUNT <newdefaultdisk>
```

The DEFAULTDISK command is not available under SDOS/MT, since the default disk is the same for all users. This is generally not a problem, since the default disk when running under SDOS/MT is normally a large capacity drive.

TIME

The TIME command is used to either display the current time of day and date, or to set the time of day and date. The form is:

```
TIME HH:MI MO/DD/YY
   or
TIME
```

The first form allows the operator to set the SDOS clock. HH is two digits which stand for hours based on a 24 hour clock (ØØ is midnight, Ø6 is 6 AM, 12 is midday, 18 is 6 PM, and 23 is 11 PM). MI is two digits standing for minutes of the hour, ranging from Ø to 59. MO is two digits for the current month, with January = Ø1, February = Ø2,... December = 12. DD is the day number within the month, 1 to 31 (you can tell SDOS that today is February 31, and it won't complain). YY is the last two digits of the year number; for 1979, it is 79.

There must be only a single space between TIME and the hours, and a single space between minutes and the month number.

When SDOS is first booted, it knows if its clock has been set (some computers keep track of time even when shut off). If the time has not been set, the command interpreter will print out the word TIME followed by a space, and expects the operator to complete it. If the operator does not complete it, SDOS will periodically pester the operator in the same fashion (this helps ensure that files get marked with their correct creation date, that reports printed are dated properly, etc.). SDOS will not allow file updates or creates when the time has not been set.

The time may be changed at will. SDOS will then accurately update the current time as time passes, adjusting the date and year (correctly) if necessary.

Example: to set the time to 3:14 PM, April 3, 1979, the operator types:

```
.TIME 15:14 4/3/79
```

Note that leading Ø digits need not be typed.

The second form of the TIME command displays the current time in the form HH:MI:SS MO/DD/YY, where SS is the current time in seconds.

```
.TIME
15:14:Ø8 Ø4/Ø3/79
```

FREE

The FREE command is used to determine how much disk space is available (unused) on a disk. The form is:

```
        FREE
```

or

```
        FREE <diskdevicename>
```

The first form tells the operator the amount of free space on the default disk, and is identical to FREE DISK:. The second form allows the operator to specify which disk is to be examined for free space.

The data displayed indicates the number of available clusters, the percentage of the disk capacity available and the number of free bytes available on the disk (note that some of the "free" bytes are used by SDOS to keep track of other bytes when a file is created, so the actual amount of storage available for data on the disk is slightly less than the value displayed).

```
        .FREE D2:
        A total of 385 free clusters for  11.7% of disk capacity
        ( 385 clusters = 591360 bytes )
```

Small disks with less than 5% free disk space probably do not have enough available space for another file.

VERSION

The VERSION command is used to determine which version of the command interpreter is actually being used, and is needed usually only if a bug is discovered. SD programs all display a banner identifying themselves when loaded; the command interpreter is the only exception. This command prints the banner for the command interpreter.

```
.VERSION
SDOS Version 1.1 SDOSCMDS V1.1k BASIC Runtime Package Version 1.4i
Copyright (C) 1977 Software Dynamics
```

This documentation matches command interpreters whose version number matches that given in the example above.

SETPROTECTION

The SETPROTECTION command is used to set a protection bit on a file, to prevent certain operations from being applied to the file. The form is:

.SETPROTECTION <letter> ON <filename>.

<letter> may be W (for Write Protect, to prevent deletion or alteration of the file), or B (for Backup Protection, to prevent redundant backing up of a file [see SDOSDISKBACKUP "CHANGED" option]). "D" stands for "Delete Protection", but is NOT implemented; use W instead.

Example:

.SETPROTECTION W ON DØ:CRUCIAL.DATA

CLEARPROTECTION

The CLEARPROTECTION command is used to remove a protection bit set by SETPROTECTION. This is useful when one wishes to delete or update a protected file. The format is:

.CLEARPROTECTION <letter> ON <filename>.

<letter> is as described under SETPROTECTION.

Example:

.CLEARPROTECTION W ON DØ:CRUCIAL.DATA

WARNING: Removing W protection from system files (i.e., *.SYS) can make the disk extremely vulnerable to operator mistakes from which recovery of data is extremely difficult.

HELP

The HELP command is used to convert an error number into a text string that people understand. Normally, this conversion is done automatically by SDOS; however, there are times when some transient hardware or software failure will prevent SDOS from printing the proper message, and it (or some program) will be forced to print a number instead. The HELP command can be used by the operator to force SDOS to again try to convert the number to its corresponding message. The form is:

```
        HELP
or
        HELP <number>
```

HELP, by itself, means "print the message for the last error number that was printed out". HELP with an explicit number, means "print the error message corresponding to this number".

```
        .HELP 1008
        No DEFAULTPROGRAM on default disk
```

If SDOS cannot convert the number to the corresponding string, it will simply print out

```
        ERROR <number>
```

as its response. This can happen if there is no ERRORMSGS.SYS file on the default disk (see DEFAULTDISK) or if the ERRORMSGS.SYS file is damaged.

Responses to certain error numbers can be somewhat ambiguous:

```
        .HELP 1045
        Disk Read Error
```

The operator cannot be sure that a disk read error really did not occur while processing the HELP command since HELP uses a disk file, ERRORMSGS.SYS.

A printed list of error messages can be found in the section on ERROR MESSAGES. Since SDOS is continually being improved, this list will continue to grow; the section on SDOSERRORMAINT shows how to get an up-to-date list.

DEBUG

The DEBUG command is used by an assembly language programmer to load a program to be tested, and pass control to the local debugger program before the program under test is executed. The form is:

```
DEBUG <filename>
```

The program specified by <filename> is loaded into the user's memory space, and control is passed to the local debugger.

A small program placed in the upper part of page zero is used to perform this process, so none of the load records may select page zero or the system may crash. The small program uses the LOAD syscall to load the program. It uses a DEBUG syscall to start the debugger, after setting the return address for the DEBUG syscall to the starting address of the loaded program.

For systems using IDB (the SD debugger), the following example is relevant:

```
.DEBUG TEST.BIN

P=2800 A=5E B=57 C=C4 X=575E S=821A */ BD0100 8887/ 00

IDB V1.1
```

The Program Counter is set to the start address specified by the load records. Single-stepping or real-time execution may be done immediately.

This command is not available under SDOS/MT.

CRC

The CRC command is used to compute a digital "signature" (termed "CRC") over the contents of a file. The signature takes the form of a (hexadecimal) number, and depends on the content and the order of information in a file. Such a signature is primarily useful when comparing two files which are purportedly identical, especially when they are not on the same computer; matching signatures indicate the files are extremely likely to be identical, while non-matching signatures indicate the files are definitely NOT identical. Very small differences make large changes in the signature of a file. The form is:

```
.CRC <filename>
```

The signature for the specified filename is computed, and printed as a hexadecimal number. Example:

```
.CRC MONTHLYDATA
CRC= :FD13
```

USERSPACE

The USERSPACE command is used to inquire how much memory space is available for user programs under a particular configuration of SDOS. The value printed is the contents of location $FC and $FD. This command is used only rarely, and generally only by assembly language programmers. Example:

```
.USERSPACE
:xxxx
```

LOG

The LOG command is used to create a copy of a terminal session, in a file or on a printing device. This is a handy way to get a hard copy of the list of files on a disk, to generate printed examples of how to use a program, and for documenting bugs in a program. The form is:

```
LOG <filename>
```

SDOS first checks to see if LOGging is already active; if so, it complains with an error message. If not, <filename> is CREATEd, and all further input and output to the operator's CONSOLE: (channel Ø) is also listed to the specified file.

```
.LOG LPT:
.FILES D2:
.
.
.
.CLOSELOG
```

is a another way of obtaining a hard copy version of the list of files on D2: (the CLOSELOG causes the listing of the console display to cease).

Killing a program (^C^C) causes logging to cease. For more detail, see SYSCALL:CREATELOG.

If a "bug" in an SD product is found, the operator should turn on logging to a hard copy device, cause the bug to be displayed on his screen, turn off logging, and send the resulting hard copy to SD. This allows SD to see precisely the circumstances under which the problem occurs and so enables us to locate the problem more quickly.

Example:

```
.LOG LPT:
.TIME <CR>                      (date it, please!)
.* BUG: DOESN'T XXXXX          (tell SD the problem
                                using comments)
. <demonstration of bug>
.CLOSELOG
```

CLOSELOG

The CLOSELOG command is used to turn logging off. To start logging again, another LOG command needs to be issued. The form is:

```
CLOSELOG
```

No parameters are required.

DO

The DO command allows the operator to specify that the contents of a file are to be used instead of keyboard entry. This is particularly useful when a sequence of commands is performed frequently. The format is:

```
DO <filename>
```

The file is OPENed, and lines read from the file are treated just as though the operator typed them on the console himself. These lines are used not only for commands to the command interpreter, but also as input for other keyboard requests by any programs that are run. Actual keyboard entry is not used until the contents of the "DO" file are completely processed.

If an error of any kind occurs while the command interpreter is executing a DO file or the command interpreter gets control because of an error, then it will "abort" the DO file and start accepting keyboard input again. The ESCape key may generally be used to cause an "Operator Requested Attention" error; if this does not work, ^C^C will always stop the DO file.

The following creates a DO file to execute a PAYROLL program, DELETE ABC.TXT, and finally execute an INVENTORY program.

```
.LIST CONSOLE: TO DOALL3.DO
PAYROLL
DELETE ABC.TXT
INVENTORY
^Z.DO DOALL3.DO          Typed by operator
.PAYROLL                 Printed by computer
<payroll executes>
.DELETE ABC.TXT          Printed by computer
.INVENTORY               Printed by computer
<inventory executes>
.                        Computer waits for operator
```

If a hard copy of the console session resulting from use of a DO file is desired, then a LOG command preceding the DO should be executed. In this case, all input is read from the DO file, and all output goes to the chosen LOG file. No display will be made on the operator's screen while some part of the DO file is left to be executed. Using the above DO file:

```
.LOG LPT:                Put hard copy of session to printer
.DO DOALL3.DO
                         <payroll executes,
                         ABC.TXT is deleted,
                         INVENTORY executes;
                         NOTE: No CONSOLE: display>
                         DO file is complete
```

DO files can only be nested one level deep, so invoking a second DO file from a first causes the command interpreter to completely forget it was processing the first DO file.

The commands LABEL, GOTO, IFERROR all are useful for doing parts of DO files conditionally.

At boot time, SDOS will automatically DO a file named "INITIALIZE.DO" if it is present on the boot disk, before asking for TIME. This is useful for configuring terminals (see SDOSSET), etc.

LABEL

The LABEL command is used to mark a target point in a DO file for GOTO and IFERROR commands, and is useful only in a DO file. The form is:

```
LABEL <labelname>
```

where <labelname> is separated from the word LABEL by a single blank, and consists of any sequence of non-blank characters. It is conventional to use only alphanumeric characters in the <labelname>. This command does nothing, and is totally ignored if the command interpreter is not searching for a LABEL as a result of a GOTO or IFERROR command.

Example:

```
LABEL THISISTHEPLACE
```

GOTO

The GOTO command is used in DO files to skip over sub-sequences of commands and data entry. It is useful only in conjunction with the IFERROR and LABEL commands. The form is:

```
GOTO <labelname>
```

where <labelname> has the form described in the LABEL command.

This command causes the command interpreter to read and ignore input lines until one is encountered of the form:

```
LABEL <labelname>
```

with a <labelname> that matches that of the GOTO. Once the matching LABEL statement is found, the command interpreter goes back into a mode of executing commands. End of file on the DO file will force the command interpreter to leave GOTO mode and resume normal operation.

If the operator ever ends up in the command interpreter (i.e., the "." prompt), and the command interpreter is echoing but ignoring everything he types, the command interpreter is probably doing a GOTO to a <labelname> it did not find. Typing ESCape or ^C^C will fix the problem.

```
.DELETE ABC.TXT          deletes ABC.TXT
.GOTO SKIPIT
.DELETE DEF.TXT          skipped by GOTO command!
.LABEL SKIPIT
.                        normal processing continues
```

IFERROR

The IFERROR command allows DO files to recover from processing errors of virtually any kind. The form is:

```
IFERROR <errornumber> <labelname>
```

where <errornumber> is an integer (any SDOS error number) and is separated from "IFERROR" and the <labelname> by at least one blank. <labelname> is the same as described for the LABEL command.

If the command interpreter regains control after an operation which did not have an error, then the IFERROR statement does nothing (is ignored).

If the command interpreter regains control immediately after an error occurs, and it is processing a DO file, then it will abort that DO file unless there is an IFERROR command that contains an <errornumber> equal to that of the actual error. The command interpreter determines this by reading commands from the DO file. If the next command is an IFERROR, the actual error number is compared to the error number embedded in the IFERROR command. If there is no match, the command interpreter reads the next command line from the DO file and processes it likewise. If the command is not an IFERROR, the DO file is aborted. This means that all the errors for which the DO file must recover must be listed, one per IFERROR statement, at the point in the DO file where the error would be detected.

If an IFERROR statement is found with a matching error number, then the command interpreter "forgives" the error, and does a GOTO to the specified label. This conditional GOTO allows blocks of commands to be conditionally skipped. Coupled with the GOTO command, virtually any error recovery may be implemented.

If, while looking for a matching IFERROR, the command interpreter encounters end-of-file on the DO file, the command interpreter forgets the error occurred and re-enters normal command interpretation mode.

The following DO file will make CURRENTDATA into BACKUPDATA:

```
* GET RID OF OLD BACKUPDATA
DELETE BACKUPDATA
IFERROR 1Ø11 WHOCARES
LABEL WHOCARES
RENAME CURRENTDATA TO BACKUPDATA
```

The IFERROR statement handles the "No such file" case by doing absolutely nothing except acknowledging the error.

The following DO file prints a MONTHLYREPORT on the line printer if the report has already been manufactured by a GENERALLEDGER program. If the report has not been manufactured, then the DO file runs the GENERALLEDGER program to obtain it, and then prints it out.

```
LIST MONTHLYREPORT TO LPT:
IFERROR 1Ø11 MAKEREPORT
GOTO DONE
LABEL MAKEREPORT
GENERALLEDGER
LIST MONTHLYREPORT TO LPT:
LABEL DONE
```

Error numbers are listed in the section on errors.

* (COMMENT)

The * command simply allows the operator to annotate what he is doing at the command interpreter level. The form is:

```
* <any text ended by <CR>>
```

The command is totally ignored; the command interpreter immediately prompts the operator for the next command.

This command finds uses in three places. First, when logging, an * command can be used to clarify what the operator is doing for a later reader of the logged file. Second, when used in a DO file, the * can act as a signal to the operator telling him what to do next (i.e., remove a disk pack from a drive, etc.), since all commands in a DO file, including the comments, are displayed as the DO file is processed (unless LOG is also used). Finally, comments may be inserted in a DO file to make it self documenting.

Example:

```
.* THIS IS A COMMENT
```

A "wait for operator" can be inserted in a DO file (usually after comments telling him what to do) by use of a

```
LIST CONSOLE:
```

command. The operator signifies he has completed the requested action by typing ^Z.

It is possible, using the "V" command of EDIT, to place a <BELL> code in a comment, so a DO file can signal an operator when it is done.

SDOSDISKINIT

SDOSDISKINIT is a utility program that converts a freshly-formatted disk into a file-structured disk suitable for storing and retrieving data from files. SDOS cannot place files on a disk until it has been SDOSDISKINITed. SDOSDISKINIT can also be used to recycle a file structured disk whose format is good, but contains no useful files. (A formatted disk is one which the disk electronics can read; most disks come formatted from the factory, but the format can be accidentally (magnetically) damaged, thus requiring re-formatting before the disk can be used. A "fresh" format can be placed on a disk by use of a FORMAT program; since FORMAT is hardware dependent, it is not part of SDOS. Refer to manufacturer's documentation).

WARNING: SDOSDISKINITing a disk containing useful data will cause loss of that data!

SDOSDISKINIT places the files BOOT.SYS, DIRECTORY.SYS, DISKMAP.SYS, BADCLUSTERS.SYS, and a vestigial SDOS.SYS on a disk. It also allows the operator to specify a disk identification, and to select a set of "tuning" parameters to optimize disk I/O for whatever applications that will be used with the disk. SDOSDISKINIT will suggest values for the tuning parameters so that the operator need not understand them to use the program.

The program is invoked by typing:

```
.SDOSDISKINIT <diskdevicename>
```

The device specified must be a disk drive. This is the drive containing the disk intended to be initialized.

The program responds with its banner,

```
*** SDOS Disk Initialization V1.1h ***
```

and then checks to make sure the disk is not already SDOSDISKINITed. If the disk is already initialized, the program will notify the operator and request verification before proceeding. Otherwise it goes on to the next question:

```
Disk id?
```

The operator must supply a 32 character maximum disk identification to be used for this disk (this identification is printed on a FILES display or at boot time). If more than 32 characters are entered, the operator is re-prompted and must type in a shorter disk identification. This disk identification may not contain a double quote (") character.

Next, SDOSDISKINIT asks:

From what disk device can the tuning parameters be copied?

There are two valid responses: a disk device name, or NONE. Giving a disk device name is the easy way out; all tuning parameters will be copied from the specified disk. Assuming another disk is mounted on the system with the proper tuning, this is the fastest and safest approach. Note that the disk drive specified should be the same type as the disk being initialized; tuning parameters for a floppy disk are not appropriate for a 10 megabyte disk and vice-versa.

If the operator supplies a disk device name, then SDOSDISKINIT will display the parameters read from the specified disk, and proceed with the disk initialization.

If NONE is specified, then the operator must supply the tuning parameters himself. The program will request: NSPC (number of sectors per cluster, the "unit" of space allocation on the disk), MINALLOC (minimum space to be allocated to a disk file when it is created), MIDALLOC ("middle" allocation, amount of space allocated to a disk file when file is extended) and MAPALGORITHM (tunes the disk for fast access). The program will recommend a value for each parameter which is reasonable. This value will be used if the operator simply types <CR> in response to the request for the parameters. All tuning parameters are determined by educated guesswork and some experimentation, although SDOSDISKINIT will make recommendations; the operator may have to SDOSDISKINIT a scratch disk several times to test various combinations of tuning parameters.

NSPC essentially determines the maximum size of a file (ignoring the physical disk limitations). If NBPS is the number of bytes per sector, the largest a file can be is (in bytes):

(NSPC*NBPS/2-1)*NSPC*NBPS

A good value to choose for NSPC is one for which files are just barely large enough to cover the entire disk. SDOSDISKINIT suggests the unique value of NSPC appropriate for this. If files on a particular disk (like an 80 megabyte storage module) do not need to be as large as the disk, then NSPC can be adjusted downward appropriately. Note that NSPC must be chosen large enough so that no more than 65534 (Logical) Clusters are required for a disk, where NLCN = INT(NSPT*NTPC*NCYL/NSPC).

If program load time is to be optimized, either choose a small value of NSPC (so that an entire cluster of the file can be loaded without dumping the header cluster of that file from the disk buffer pool) or a large value (so that the header cluster is not often needed, and so the cost to refetch it to the pool is negligible).

MINALLOC determines the minimum disk space (in clusters) allocated to a new disk file for this disk. Making MINALLOC large decreases the frequency with which the allocator must add more disk space to a disk; it is a good idea to set MINALLOC to a value slightly larger than the average number of data clusters in small file (SDOS always allocates at least the header cluster of the file if the file is not contiguous). "1" should be used if no better value is obvious (Ø is not a valid MINALLOC).

MIDALLOC determines how much disk space is to be incrementally added to a file when it needs to grow. This value must be larger than zero! Use 1 if no better value is obvious.

MAPALGORITHM is used to tune rotational and seek times. Actual interpretation of the 16 bit number given here is completely up to the disk-sector driver for the disk device which contains the disk being initialized. Commonly, the 16 bits are treated as a cylinder-to-cylinder seek "spiralling" byte and a sector "spacing" byte (most and least significant, respectively). The "spiralling" byte defines the delay (measured in units of sector times) between the last logical sector on track T and the first logical sector on track T+1, and is tuned best when it is just larger than the actual track-to-track seek time (this may also require accounting for head settling time). The sector "spacing" specifies the distance in time between one logical sector and the next (within a track); 1 indicates the LSNs are physically adjacent. Most SDOS systems cannot pick up two adjacent LSNs unless they are spaced every other sector (spacing = 2); optimal spacing is just slightly longer than the minimum amount of time to read a sector +6ms. The MAPALGORITHM is usually printed and entered as a hex number (:xxyy). Once the Mapalgorithm has been chosen, SDOSDISKINIT proceeds with the disk initialization.

When the program asks for a mapalgorithm, the user may enter <CR>, which selects a default of :ØØØ1 (slow, but always works), he may enter a mapalgorithm number known to be appropriate for the type of disk being initialized, or he may ask SDOSDISKINIT to recommend a mapalgorithm by entering "?<CR>". When SDOSDISKINIT is asked to make a recommendation, it tests various mapalgorithms (using the specified drive) to determine which mapalgorithm works fastest. This process takes about half an hour, so it isn't done very often; one should record the resulting mapalgorithm number so repeating this process isn't necessary. Because different programs operate best at different disk speeds, the suggested map algorithm might be a little too fast; another mapalgorithm to try is the suggested one, plus :1Ø1 (i.e., :4Ø7 plus :1Ø1 is :5Ø8). Only experimentation will tell.

A final tuning parameter, the expected number of files for this disk, must also be entered. SDOSDISKINIT will supply a hint, based on an average file size of 8192 bytes. The value entered does not limit the number of files on a disk; it is used merely to pre-allocate enough DIRECTORY.SYS space to allow quick location of file names over the life of the disk (without having to expand the directory; such expansion ruins all the hashing).

SDOSDISKINIT will make a bootable system disk if desired by copying BOOT.SYS, SDOS.SYS, SERIALNUMBER.SYS and DEFAULTPROGRAM from another system disk if so requested (BOOT.SYS and SDOS.SYS contents cannot be copied correctly by COPY or SDOSDISKBACKUP). Other files desirable on a system disk (such as ERRORMSGS.SYS, utilities, etc.) must be copied by the COPY or SDOSDISKBACKUP command to the newly initialized disk. If only a data disk is desired, the operator should respond NO or <CR> when SDOSDISKINIT asks if the disk should be a system disk.

Notes about operation of SDOSDISKINIT:

1) SDOSDISKINITing the disk in the default drive works, but leaves SDOS with a dilemma on completion; from where can it get a DEFAULTPROGRAM? For this reason, it is not recommended to perform an SDOSDISKINIT on the default drive.

2) The operator should insure that write protect on the chosen drive is disabled, that the inserted disk has already been formatted, and that TIME has been set before starting an SDOSDISKINIT.

3) Installation of SDOS.SYS on a disk (via SDOSSYSGEN) should be performed only immediately after that disk has been SDOSDISKINITed.

Examples:

```
.* User wants to initialize a data disk to be just like another
.SDOSDISKINIT D2:
*** SDOS Disk Initialization V1.1h ***
Disk id? Data Disk
From what disk device can the tuning parameters be copied? DØ:
NBPS= 256  NLSN= 1232  NLCN= 2Ø5  NSPC= 6  Map algorithm=:ØØØ6
How many files do you anticipate having on this disk?
(Default value is 38 ):
Is this to be a bootable system disk (Default = NO)?
Disk initialization complete.


.* User wants to set tuning parameters
.SDOSDISKINIT D2:
*** SDOS Disk Initialization V1.1h ***
Disk id? Data Disk Tuned for Payroll
From what disk device can the tuning parameters be copied?
Disk has  315392 bytes.
How many sectors per cluster (Default value is 4 ) 4
Minimum allocation (Default value=1): 3
Minimum extension (Default value=1): 1Ø
Map Algorithm (Default value :ØØØ1, ? to find best): :Ø2Ø5
NBPS= 256  NLSN= 1232  NLCN= 3Ø8  NSPC= 4  Map algorithm=:Ø2Ø5
How many files do you anticipate having on this disk?
(Default value is 38 ): 6Ø
Is this to be a bootable system disk (Default = NO)? NO
Disk initialization complete.


.* User wants to build a bootable system disk, using DØ: as a model
.SDOSDISKINIT D2:
*** SDOS Disk Initialization V1.1h ***
Disk id? System Disk
From what disk device can the tuning parameters be copied? DØ:
NBPS= 256  NLSN= 1232  NLCN= 2Ø5  NSPC= 6  Map algorithm=:ØØØ6
How many files do you anticipate having on this disk?
(Default value is 38 ): 38
Is this to be a bootable system disk (Default = NO)? YES
File to copy into BOOT.SYS (Default = None)? DØ:BOOT.SYS
File to copy into SDOS.SYS (Default = None)? DØ:SDOS.SYS
File to copy for SERIALNUMBER.SYS (Default = None)? DØ:SERIALNUMBER.SYS
File to copy for DEFAULTPROGRAM (Default = None)? DØ:DEFAULTPROGRAM
Disk initialization complete.
```

```
.* User wants to initialize a data disk with
.* the best possible map algorithm
.SDOSDISKINIT D2:
*** SDOS Disk Initialization V1.1h ***
Disk id? Programs Disk
From what disk device can the tuning parameters be copied? NONE
Disk has  315392 bytes.
How many sectors per cluster (Default value is 4 )
Minimum allocation (Default value=1): 1
Minimum extension (Default value=1): 10
Map Algorithm (Default value :0001, ? to find best): ?
:0001,  30.80 sec,   1461 bytes/sec
:0002,  32.63 sec,   1378 bytes/sec
:0003,  34.40 sec,   1308 bytes/sec
:0004,  31.26 sec,   1439 bytes/sec
:0005,  12.63 sec,   3562 bytes/sec
:0006,  13.96 sec,   3221 bytes/sec
:0007,  14.26 sec,   3154 bytes/sec
:0008,  16.43 sec,   2738 bytes/sec
:0009,  16.53 sec,   2721 bytes/sec
:000A,  18.30 sec,   2459 bytes/sec
:000B,  20.30 sec,   2216 bytes/sec
:000C,  21.93 sec,   2051 bytes/sec
:000D,  23.63 sec,   1904 bytes/sec
:000E,  25.46 sec,   1767 bytes/sec
:000F,  27.23 sec,   1652 bytes/sec
:0105,  13.76 sec,   3268 bytes/sec
:0205,  12.53 sec,   3590 bytes/sec
:0305,  12.30 sec,   3658 bytes/sec
:0405,  12.90 sec,   3488 bytes/sec
:0505,  13.00 sec,   3461 bytes/sec
:0605,  12.13 sec,   3708 bytes/sec
:0705,  12.20 sec,   3688 bytes/sec
:0805,  12.93 sec,   3479 bytes/sec
:0905,  12.53 sec,   3590 bytes/sec
:0A05,  12.50 sec,   3600 bytes/sec
:0B05,  12.13 sec,   3708 bytes/sec
:0C05,  13.53 sec,   3325 bytes/sec
:0D05,  13.46 sec,   3341 bytes/sec
:0E05,  13.06 sec,   3443 bytes/sec
:0F05,  13.16 sec,   3417 bytes/sec
Map algorithm (Default = :0605):
NBPS= 256  NLSN= 1232  NLCN= 308  NSPC= 4  Map algorithm=:0605
How many files do you anticipate having on this disk?
(Default value is 38 ):
Is this to be a bootable system disk (Default = NO)?
Disk initialization complete.
```

SDOSDISKBACKUP

The SDOSDISKBACKUP program is used to copy the entire contents of one disk onto another disk, or to copy a subset of the files on one disk to another disk. This is primarily useful when making "backup" copies of disks or files, to aid simple recovery from accidental future destruction of such disks or files.

SDOSDISKBACKUP can be told to selectively copy a disk, to copy files by name, by wildcard, by date limits (BEFORE/AFTER), if the file has not been backed up (CHANGED), or is NOT the member of a list of files. Because a file might be too large to store (on the remaining space on) on a single disk, a facility to split large files while backing up is provided.

Space for the copied file is pre-allocated (unless the file is sparse) to ensure that backed-up files are as "contiguous" as possible, enhancing sequential access to backed up copies of a file.

After making copies, the program verifies that each copy made is indeed a duplicate of the original file, thus providing protection against disk controller failures, memory faults in the computer, or software bugs. Thus, SDOSDISKBACKUP is preferred over COPY for making copies of files when absolute gaurantees of identical copies is required.

The SDOSDISKBACKUP program also provides mechanisms for copying in the presence of (disk) I/O errors, thus providing a way of saving "most" of file if a critical copy is damaged.

Since backing up a set of files or a disk can be fairly time consuming, or complicated by I/O errors, a NONSTOP mode is provided under which a "best effort" is made by SDOSDISKBACKUP before it automatically proceeds to skip around the cause of an error, usually by skipping a portion of the file being copied.

Sometimes, the MAPALGORITHM on a disk needs to be changed to enhance performance of the system when using the contents of that disk. SDOSDISKBACKUP can change the Map Algorithm when copying to a disk. This facility is generally only used by experts.

The command format for SDOSDISKBACKUP is:

SDOSDISKBACKUP <source> {<qualifiers>} {NONSTOP} TO/OVER <destination>

When backing up a single file, the command format is identical to that of the COPY command (to prevent confusion). However, when one is not making an archival copy of a single file, the conventional COPY command is more suitable, since it is faster than SDOSDISKBACKUP.

The <source> specifies which files are to be copied. <Qualifiers> specify extra conditions which must be met by each file in the source list in order for that a file to be copied. NONSTOP indicates a "best effort" is required, but that SDOSDISKBACKUP should proceed automatically in event of a failure; absence of NONSTOP causes SDOSDISKBACKUP to query the operator about how to recover from a failure. OVER indicates that a file should be deleted from the destination before it is copied from the source; TO causes a complaint to be issued if the file is present at the destination before it is copied, preventing accidental overwrite of an already backed-up file. <destination> indicates to where the files must be copied, and (sometimes) optional parameters about the destination disk.

The <source> may be the name of a disk device, whose content is to be copied, or it may be a list, separated by commas, of standard SDOS file names, wildcard file names, or file names preceded with the character "@" (indirect filenames). Mention of a file name in the list causes the file to be copied, provided that the specified qualifier conditions are met. All files copied must be from the same disk; if a disk name is given, it must be the leftmost item in the list.

A wildcard file name causes all qualified files whose name matches the wildcard (see FILES command) to be copied. A filename preceded by an "@" (AT sign) indicates that the file specified contains a list of file names or wildcards, to be treated as above; nested AT files are not allowed.

Qualifiers are optional; if they are given, they may come in any order. The allowable qualifiers are: EXCEPT <list>, CHANGED, AFTER <date> and BEFORE <date>. If multiple qualifiers are given, only files which match the conditions specified by all the qualifiers are copied.

EXCEPT allows specification of list of (wildcard or indirect) file names NOT to be copied; a "?" may be included as an element of the list, which will cause SDOSDISKBACKUP to ask the user about backing up qualified files which match wildcards in the source list, so the user may perform a selective backup.

CHANGED indicates that only files whose Backup (B) protect bit is clear are to be copied; files backed up (without error) then have their Backup protection bit set (SDOS clears the Backup protection on a file whenever a file is created or updated).

AFTER <date> indicates that only files whose creation or last update date (see FILES command) follows the specified date are to be copied. The date is specified as MM/DD/YY, with MM equal to Month number, DD is a Day number, and YY is the rightmost 2 digits of the year number.

BEFORE <date> indicates that only files whose date is before the specified date are to be copied. The <date> has the same form as in the AFTER qualifier. Selecting both the AFTER and BEFORE options allows copying of files which were created/updated within a certain period of time.

NONSTOP indicates that errors are to be reported but ignored; a best effort copy is all that is desired. If an error occurs, the user is notified as to the type of error and its location in the source/destination files. SDOSDISKBACKUP will do the least-destructive recovery it can to get around the error; in the case of Disk Read or Write errors, this generally means skipping a sector, so that the backed-up copy is damaged in only the sector that gives trouble. If an SDOSDISKBACKUP command is being executed from a DO file, NONSTOP mode is automatically assumed. Failure to complete backup perfectly will cause Error 104, which will abort a DO file unless error recovery (see IFERROR) is installed. If the backup is not run in nonstop mode, and an error occurs which can be recovered from perfectly, no Error 104 will be issued.

TO specifies that the files are to be copied to the destination; complaints will be issued if a file by the same name as one to be copied exists on the destination, and the file will not be copied unless the user authorizes the program to do so (note: the COPY command in the command interpreter does NOT make this check). OVER specifies that any file on the destination with the same name as a file to be copied is to be replaced, by deleting the file from the destination before the copying process starts.

INTO is a special form of TO, which specifies that if the destination file exists, it is to be OPENed and set to zero filesize instead of CREATEd. This allows special system files, such as SDOS.SYS and BOOT.SYS, to be filled via SDOSDISKBACKUP. If the destination file does not exist, INTO is treated identically to TO. SDOSDISKBACKUP will tell the user when it is copying whether TO, INTO, or OVER is required for each individual destination file.

The <destination> specifies where the <source> file(s) is to be copied. If the destination specified is simply "*" (asterisk) or a disk device name followed by an "*", then files are copied from the source to the destination using the source names (this is compatible with the COPY command). If the destination is a single file name or a disk device name followed by a single file name, then the source must be a a single file, and is copied to the destination using the new name (also compatible with COPY).

If the destination is simply a disk device name, then the <source> must specify only a single file or disk; the contents of the source are then copied onto the contents of the destination. This is normally used only when making copies of complete disks, file system and all, as it destroys the original contents of the destination disk. SDOSDISKBACKUP requires verification from the operator in this case to prevent accidental destruction of disk contents.

An optional phrase, USING MAPALGORITHM :xxxx, may be appended to the disk specified as the destination. This causes SDOSDISKBACKUP to set the Map Algorithm on the destination disk to the specified value. SDOSDISKBACKUP cannot change the mapalgorithm of a disk by copying it onto itself. See the table below for description of how Map Algorithms are set/used by SDOSDISKBACKUP when disk devices are involved. For examples of usage with floppies and Winchesters, see the discussion under the COPY command.

This page can be skipped if you are not an expert.

The following table describes in detail what happens for each of the various <source> and <destination> combinations. "wildcardspec" means a list of wildcard file names, AT file names, or conventional file names.

Actions of SDOSDISKBACKUP with respect to Map Algorithms

source spec	destination spec	src disk note	dst disk note	legal	map
!diskdevice:	diskdevice:	! 2	! 3	! yes	!yes!
!diskdevice:	*	!	!	! no	! x !
!diskdevice:	singlefile	! 1	!	! yes	! no!
!singlefile	diskdevice:	!	! 4	! yes	!yes!
!singlefile	*	!	!	! yes	! no!
!singlefile	singlefile	!	!	! yes	! no!
!wildcardspec	diskdevice:	! 5	!	! no	! x !
!wildcardspec	*	!	!	! yes	! no!
!wildcardspec	singlefile	! 5	!	! no	! x !

Notes:

1. This generates an image of the disk.
   If not an SDOS disk (error when open 'boot.sys')
   then set Map Algorithm to :0001
   else use Map Algorithm given by source disk

2. If not an SDOS disk (error when open 'boot.sys')
   then set map to :0003 (this is a hueristic that works well)
   else set Map Algorithm to that of disk

3. If USING MAPALGORITHM specified
   then use specified Map Algorithm
   else use same Map Algorithm as source disk

4. If USING MAPALGORITHM specified
   then use specified Map Algorithm
   else
       if BOOT:CHECKSUM is correct in the source file
       then use Map Algorithm from 'boot.sys' in the source file
       else use Map Algorithm :0001 (file is not image of SDOS disk)
   fi

5. Give error 'Concatenation copies not allowed'
   (SDOSDISKBACKUP does not concatenate multiple files when copying).

Some operator intervention is required when running the program normally (NONSTOP option not specified). If the destination specified was a disk, the operator must respond "YES" to the query about overwriting the destination disk (note: if you do not understand what this means, type "NO" when asked or you will probably be extremely sorry). If "?" was specified in the EXCEPT list, SDOSDISKBACKUP will query the operator only for files which match the qualifiers; a "N" or "NO" response will prevent the file from being copied, and any other response (including just <CR>) is interpreted as "YES".

Should there be no room remaining on the destination disk when backing up files, SDOSDISKBACKUP will notify the operator, and request one of three actions: give up trying to copy the file altogether (ABORT), remove the copied fragment from the destination disk, and copy the fragment and the balance of the file to a newly specified destination disk (MOVE), or leave the copied fragment on the destination, and copy the balance of the file to a new destination (SPLIT). Since backing up files requires that the destination disk be SDOSDISKINIT'ed, if one intends to back up a lot of files to a floppy, or some file may be split because it is too big for a floppy, it is a good idea to SDOSDISKINIT several disks before starting the backup, so they may be substituted as necessary. SDOSDISKBACKUP can also split a file being moved onto a non-file structured disk device.

Specifying ABORT causes SDOSDISKBACKUP to give up trying to copy this particular file. The fragment that filled up the destination disk is deleted.

If MOVE is specified, SDOSDISKBACKUP first deletes the fragment copied onto the current destination disk. It then asks the user for the name of the device containing a disk on which the backup is to continue, and a filename to be used as the name for the balance of the file being backed up. Entering just <CR> causes SDOSDISKBACKUP to use the same disk device, and the same file name (this is useful when a new diskette will be substituted in the same drive); otherwise the user must specify both the new destination drive name and file name. Once the new destination has been specified, SDOSDISKBACKUP then dismounts the original destination disk, and notifies the user; the user MUST NOT remove the original destination disk until the notification of dismount has been printed. After the dismount message, the new disk may be inserted; SDOSDISKBACKUP will wait for the user to enter <CR> before proceeding with the backup.

If SPLIT is specified, SDOSDISKBACKUP leaves the fragment copied onto the current destination disk intact. It then asks the user for the name of the device containing a disk on which the backup is to continue, and a filename to be used as the name for the balance of the file being backed up. Entering just <CR> causes SDOSDISKBACKUP to use the same disk device, and an "incremented" version of the same file name (incremented names are discussed further below); otherwise, the user must specify both the new destination drive name and file name. Once the new destination has been specified, SDOSDISKBACKUP then dismounts the original destination disk, and notifies the user; the user MUST NOT remove the original destination disk until the notification of dismount has been printed. After the dismount message, the new disk may be inserted; SDOSDISKBACKUP will wait for the user to enter <CR> before proceeding with the backup.

If a file name has the form <name.nnn>, the incremented form is <name.nnn+1>, i.e., FRED "incremented" is FRED.1, and FRED.1 incremented is FRED.2, etc. Using the default incremented names allows a large file to be conveniently split into several pieces, the first of which is named with the original file name, and the remaining pieces are sequentially named. The COPY command can be used to regenerate the original file from the pieces.

Disk I/O errors are handled by SDOSDISKBACKUP. In NONSTOP mode, a disk I/O error when copying a file causes that sector of the file to simply be skipped. A disk error when opening a file causes that file to be be skipped. A disk error while scanning the directory for a match on a wildcard will cause all files in that sector of the directory to be skipped. In non-NONSTOP mode, disk I/O errors during the copy process cause SDOSDISKBACKUP to query the operator as to whether to ABORT the file copy, to RETRY reading the offending sector, or to SKIP the sector and continue copying. Some classes of damaged disks are best handled by copying the entire disk in NONSTOP mode onto another, SDOSDISKVALIDATEing the copy, and then copying files off the duplicate.

Examples of use:

```
.SDOSDISKBACKUP DØ:FRED TO D1:FRED
```

copies FRED from DØ: to D1: This is effectively identical to COPY FRED TO D1:FRED, except SDOSDISKBACKUP will complain if FRED is already present on D1: (and then provide an opportunity to do it anyway).

```
.SDOSDISKBACKUP FRED TO D1:*
```

copies FRED from DISK: to D1:.

```
.SDOSDISKBACKUP FRED OVER D1:*
```

copies FRED from DISK: to D1:. If a file named FRED was already present on D1:, it is deleted from D1: before FRED is copied from DISK:. Typing SDOSDISKBACKUP FRED OVER FRED simply does nothing, which gives the desired effect.

```
.SDOSDISKBACKUP FRED TO FØ:SAM
```

copies FRED from DISK: to FØ:, renaming it SAM in the process.

```
.SDOSDISKBACKUP *.BAS,FRED TO D3:*
```

copies all files whose extension is .BAS, and the file FRED, to D3:.

```
.SDOSDISKBACKUP WDØ:* CHANGED TO FØ:*
```

copies all files which have been created or changed since the last time a SDOSDISKBACKUP WDØ:* CHANGED ... was performed, to FØ:. This is one of the most common forms of use, especially when WDØ: is a Winchester drive, and FØ: is a floppy. Files backed up are then marked as "Backup Protected" (the "B" in a FILES command under the Protection column) to prevent backing them up again unless they are changed (the B bit is cleared when a file is created or updated).

```
.SDOSDISKBACKUP *.BAS,*.TYP EXCEPT JUNK*,? CHANGED TO D1:*
*** SDOSDISKBACKUP V1.1g ***
Copy DISK:REJECT.TYP to D1:REJECT.TYP? (<CR>=yes)
Copy DISK:IS.TYP to D1:IS.TYP? (<CR>=yes) NO
Copy DISK:DISKVAL411H.BAS to D1:DISKVAL411H.BAS? (<CR>=yes)
```

copies all files whose extension is .BAS or .TYP, which do not start with JUNK..., and have been changed, to D1: Because "?" appeared in the EXCEPT list, the SDOSDISKBACKUP program asks the operator about whether to back up each file, individually, which meets these criteria.

```
.SDOSDISKBACKUP *.ASM AFTER 2/28/83 BEFORE 7/1/83 TO WDØ:*
```

copies all .ASM files from DISK: to WDØ: provided they were created after February and before July.

```
.SDOSDISKBACKUP DØ: TO D1:
*** SDOSDISKBACKUP V1.1g ***
Writing on the DISK device can damage the file structure.
Are you sure you want to write on the DISK device? YES
Copying DØ: to D1:
```

causes the entire contents of DØ: to be copied to D1:. This form is especially useful when making an archival copy of system or boot disks. It is generally sensible only when DØ: and D1: are the same class and size of disk device.

```
.SDOSDISKBACKUP DØ: TO D1: USING MAPALGORITHM :3Ø2
*** SDOSDISKBACKUP V1.1g ***
Writing on the DISK device can damage the file structure.
Are you sure you want to write on the DISK device? YES
Copying DØ: to D1:
```

is useful when DØ: does not have Map Algorithm :3Ø2, and it is known that :3Ø2 will make the system run better when using that disk (the value :3Ø2 must be determined empirically; SDOSDISKINIT can make a good suggestion).

```
.SDOSDISKBACKUP DØ: TO WDØ:DISKIMAGE
```

can be used to save a copy of the entire contents of a floppy disk DØ: in a file on a Winchester disk WDØ:. This is very useful on systems that have a single floppy drive and a Winchester disk drive.

```
.SDOSDISKBACKUP WDØ:DISKIMAGE TO DØ:
*** SDOSDISKBACKUP V1.1g ***
Writing on the DISK device can damage the file structure.
Are you sure you want to write on the DISK device? YES
Copying WDØ:DISKIMAGE to DØ:
```

can be used (after the preceeding step) to make a duplicate of a saved floppy disk image onto another floppy.

```
.SDOSDISKBACKUP WDØ:DISKIMAGE TO DØ: USING MAPALGORITHM :3Ø2
*** SDOSDISKBACKUP V1.1g ***
Writing on the DISK device can damage the file structure.
Are you sure you want to write on the DISK device? YES
Copying WDØ:DISKIMAGE to DØ:
```

makes a new floppy from a disk image with a different Map Algorithm.

Here's an example of how to backup a large file that requires SPLITting. In the example, the first disk did not have enough room to be satisfactory, so the copied fragment was MOVEd.

```
.SDOSDISKBACKUP WDØ:LARGEFILE TO DØ:*
*** SDOSDISKBACKUP V1.1g ***
Copying WDØ:LARGEFILE to DØ:LARGEFILE
**** 'Disk space exhausted' occured while attempting to
write the destination file @ 3Ø7
...Successfully copied 3Ø7 bytes (3.68%) of the source file
...The destination file contains 3Ø7 bytes (3.68%) of the source file

There are three choices:

1. Move the portion of the output file already copied from
   this disk onto another disk and continue copying (option
   M).
2. Split the output file by keeping the portion of the output
   file already copied on this disk, and continue copying
   (option S). Split is not allowed if source disk and
   destination disk are the same.
3. Abort this file copy and delete the portion of the output
   file already copied (option A).
Move, Split, or Abort (M, S, A, <CR>=Split)? M
...DØ:LARGEFILE deleted
New destination file name (Default=DØ:LARGEFILE)?
DØ: dismounted
<CR> when new destination device is ready
**** 'Disk space exhausted' occured while attempting to
write the destination file @ 3Ø72
...Successfully copied 3Ø72 bytes (36.8%) of the source file
...The destination file contains 3Ø72 bytes (36.8%) of the source file

There are three choices:

1. Move the portion of the output file already copied from
   this disk onto another disk and continue copying (option
   M).
2. Split the output file by keeping the portion of the output
   file already copied on this disk, and continue copying
   (option S). Split is not allowed if source disk and
   destination disk are the same.
3. Abort this file copy and delete the portion of the output
   file already copied (option A).
Move, Split, or Abort (M, S, A, <CR>=Split)? S
New destination file name (Default=DØ:LARGEFILE.1)?
DØ: dismounted
<CR> when new destination device is ready
.* BACKUP has completed
```

SDCOPY

The SDCOPY command is used to make exact copies of disks, or to copy a file from one disk to a different disk, using only a single disk drive. This program should only be used if the computer system has just one disk drive; otherwise, the COPY command is more convenient. Making a copy of a file under a different name on the same disk is also best done by COPY.

The form of the SDCOPY command is:

```
SDCOPY <sourcefile> TO <destfile>
     or
SDCOPY
```

The first form is the terse form; the program will copy the contents of the specified <sourcefile> into a file whose name is <destfile>. <destfile> may be single asterisk ('*'); this is shorthand for "use the same name as <sourcefile>". The second form simply prompts the operator for the name of the <sourcefile>, and then the name of the <destfile>; further processing is identical to that of the first form.

Once the <sourcefile> and <destfile> have been selected, the program starts the copy. Since there is only a single disk drive involved in the copying process, manual switching between two disks may be required; the program will prompt the operator with a message to insert the appropriate disk at the appropriate time, and to depress the <CR> key when the appropriate disk has been entered. For large files, this switching process may occur many times in succession. Once the copying process is complete, SDCOPY will ask the operator to insert a system disk (one with DEFAULTPROGRAM on it), and press <CR>; this exits the program.

It is very important for the operator to keep careful track of the source and destination disks, and place them in the correct order into the drives, or the copy process will fail; such failure can cause source file to be destroyed if the <sourcefile> and <destfile> have the same name! A recommended practice when using SDCOPY is to ensure the that disk containing the source file is write-protected while performing the copy.

Note that since SDCOPY performs a DISMOUNT between each disk switch, SDCOPY can copy a file from any disk to any other disk; it is not limited to copying files from the disk on which the SDCOPY program resides.

Example of file copy. Each time the program requests "Insert ... disk", the operator must insert the appropriate disk, and then press <RETURN>. The source disk is the disk on which <sourcefile> resides. The target disk is the disk where the <destfile> is to be placed. In this example, the operator did not have to switch disks the first time, because FRED.TXT was on the same disk as SDCOPY, but he did have to press <RETURN>. The operator did switch disks a total of 4 times; the last time, the original (source) disk was inserted.

```
.SDCOPY
Single Disk Copy v1.xx
Name of Source File:  FRED.TXT
Name of Target File:  SAM.TXT
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
COPY COMPLETE: Insert System Disk, hit <RETURN>
.
```

Here is an example of a file copy to a file of the same name. The source file is NOT on the same disk as SDCOPY, so a switch must be performed the first time "Insert Source Disk..." is requested. The disk containing SDCOPY must be inserted when "Insert System Disk..." is requested.

```
.SDCOPY FRED.TXT TO *
Single Disk Copy v1.xx
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
       ...
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
COPY COMPLETE: Insert System Disk, hit <RETURN>
.
```

SDCOPY will copy an entire disk onto another disk if DISK: is used for both <sourcefile> and <destfile> (when copying a file, device names may NOT be used). This is especially appropriate when one wishes to make a duplicate of the SDOS system boot disk. SDCOPY will ask the operator if a disk copy is really what was desired, and will proceed only on a YES response (using a disk copy when a file copy is intended will destroy the entire contents of the destination disk). The procedure for copying a disk requires the same switching process as for copying files, but generally takes more switches to accomplish.

Since SDCOPY performs a DISMOUNT between each disk switch, SDCOPY can copy any disk to any other disk; it is not limited to copying the disk on which the SDCOPY program resides.

Here is an example of an entire disk copy.

```
.SDCOPY DISK: TO DISK:
Single Disk Copy v1.xx
Are you sure you want to write on the disk DEVICE? YES
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
       ...
Insert Source Disk, hit <RETURN>
Insert Target Disk, hit <RETURN>
COPY COMPLETE: Insert System Disk, hit <RETURN>
```

SDOS USER'S MANUAL
SECTION XII: USER PROGRAMS

This section gives an extremely brief description of how to start programs commonly used under SDOS. More details can be found in the manuals describing the programs themselves.

EDIT

The SD Text EDITor is used to edit and correct files containing text, in a line-oriented style. It is started by typing its name:

```
.EDIT
```

It will identify itself,

```
EDIT V1.1x Copyright (C) 1979 Software Dynamics
```

and then accept commands. The user should select an input file (using ER) and an output file (using EW), or a combination (using EB), and then begin editing.

```
*EBPRIMES1.BAS\
```

The line following the word EDIT is passed to the editor as commands; a quick edit can be done via:

```
.EDIT EBPRIMES.BAS\1ACHELLO\BYE\EXIT<CR>
.
```

SEDIT

The SD Screen Editor, SEDIT, is used to edit and correct text files using a full-screen display. It is started by typing:

```
.SEDIT <filename>
```

If the file did not previously exist, the operator must tell SEDIT to "create" the file.

ASM

The SD Assembler is used to convert symbolic machine language into directly executable machine code. It is started by typing its name:

```
.ASM
```

It will identify itself, and then ask for Source, Listing, and Binary files.

```
SOFTWARE DYNAMICS MAL/680x, Version 1.x/ xxxx
Source File=DRIVER.ASM
Listing File=LPT:
Binary File=DRIVER.BIN
>
```

BASIC

The SD BASIC Compiler is used to convert BASIC program source files to assembly source files. It is started by typing its name:

```
.BASIC
```

It will identify itself, and then ask for the name of the file to be compiled, and the name of an output file.

```
.BASIC
Software Dynamics BASIC Compiler Version 1.4h1 (C) 198Ø
Input file = PRIMES.BAS
Output File = JUNK
Compilation Complete
```

The output of the Compiler needs to be assembled. The assembled output may be executed by simply typing its name to the command interpreter. Typically, FIX or COMPILE is used instead of running BASIC directly.

COMPILE

COMPILE is used to compile and assemble a BASIC program (this is easier than invoking the BASIC compiler and Assembler individually). It is invoked by typing:

```
.COMPILE MYPROG.BAS
```

A BASIC Compile and assemble are automatically executed.

FIX

FIX is used to EDIT and COMPILE a BASIC program. It is invoked by typing:

```
.FIX ABC.BAS
```

(the extension .BAS is important!) FIX will cause the current copy of ABC.BAS to be renamed to ABC.BAK (so a backup "copy" is made), and then ABC.BAK is "EDIT"ed into ABC.BAS (FIX will invoke the EDITor automatically). When EXIT is typed to the EDITor, COMPILE is automatically invoked.

SHUTTING DOWN

To stop operation of the computer under SDOS, the operator needs to DISMOUNT all the disk devices (see DISMOUNT command of SDOSCOMMANDS). This cannot be (conveniently) done while a program other than SDOSCOMMANDS is running, so the operator must wait for any currently running programs to stop execution or manually kill program execution via ^C^C. A turnkey system will have special provisions for shutting down and performing the required DISMOUNTS.

The dismount commands are given to SDOSCOMMANDS. FAILURE TO DISMOUNT A DISK WHICH WAS USED IN ANY WAY BY SDOS CAN MEAN LOST DATA (SDOS works hard to prevent this even if DISMOUNTs are not used, but the scheme is not perfect).

After dismounting all disk drives, the operator may push RESET to cause execution of SDOS to cease.

This shutting down procedure must also be performed if it is desired to boot from another disk cartridge or to obtain error messages from an ERRORMSGS.SYS file on a drive other than the one booted from.

If an Error 1046 (Disk Write) occurs during a DISMOUNT, your disk is probably software damaged; certainly, some data has been lost. Run SDOSDISKVALIDATE immediately (before attempting to DISMOUNT again). If another Disk Write error occurs while starting SDOSDISKVALIDATE, just start it again. Keep trying until the SDOSDISKVALIDATE succeeds in repairing your disk.

An Error 1002 (File is Open) during a DISMOUNT indicates SDOS is a little confused (this happens very rarely, usually due to an I/O package bug); the DISMOUNT was successful in moving all your data back to the disk, but SDOS has its fingers glued to the disk. Ignore the message; but you must run SDOSDISKVALIDATE the next time you get a chance.

If there is a malfunction, and the operator cannot get SDOS to respond in any way (see ^C^C), then SDOS has died and effectively shut down. This kind of "shutdown" will very probably software damage all the disks in use at the time of the failure; use the SDOSDISKVALIDATE program on all of them.

When shutting down SDOS/MT system, one must first ensure that all users are off the system, and then dismount all disks before doing a RESET.

DISASTERS, or "What do I do if..."

There are many circumstances in which SDOS might not behave as expected. Many times this can happen due to ignorance of the operator of some apparently minor detail, so unexpected operation is not always a disaster. This section describes some of the more common disasters that occur, and gives some hints on how to deal with them.

1) "SDOS won't boot". Guess bad hardware, not a system pack, or a damaged system pack. See BOOTING SDOS.

2) "SDOS boots, but won't run my program". Your program might not be on the particular disk specified ("No Such File" will appear as the error message); perhaps it is on a disk in another drive. Use the

   FILES Dn:myprog

   command to see if SDOS has that program in its directory on each disk "n"; try all the disks. If not, you will have to get a copy of the program before you can run it.

3) "It didn't complain when I asked it to run my program, but nothing seems to be happening". If any disk activity is occurring, just be a little patient; if typing ^C <LF> echoes ^C, things are probably OK (if no ^C is echoed, SDOS is quite dead and you need to re-boot). If nothing is happening at all, perhaps the program is waiting for input; since most input to SDOS or application programs is done on a line-by-line basis, perhaps typing <CR> will help. If there is still no response, try ^C^C. This will cause SDOS to kill the program, and return control to DEFAULTPROGRAM. Killing the program may cause it to leave disk data files contents a little confused, so don't do this unless you're convinced something is wrong. If you have to kill an application program, it probably has a bug in it; tell somebody. If ^C^C obtains no response, but does echo, and nothing happens, then there is also probably a bug; you will have to RESET and re-boot; run the SDOSDISKVALIDATE program.

4) "It said, 'Disk Read (Write) Error'". The file SDOS was reading is partially damaged, and some data was lost. Use the SDOSDISKBACKUP program to copy the file (read errors cause COPY to quit); then run the SDOSDISKVALIDATE program. Finally, the contents of the data file need to be reviewed since at least one sector's worth has been damaged. SDOSDISKVALIDATE will find the cluster which caused the error and add it to BADCLUSTERS.SYS so it cannot cause any more trouble. NOTE: FAILURE TO RUN SDOSDISKVALIDATE AND TO CLEAR UP ANY DISK FILE STRUCTURE ERRORS MAY RESULT IN A FILE SYSTEM DISASTER AT A LATER TIME!

5) "It said, 'Disk Full'". Delete some old files you are not going to need anymore and try again.

6) "SDOS Checksum failure". Either memory is beginning to fail, or (more likely), the application program just run has a bug in it which steps on part of SDOS. You need to re-boot; if you were not debugging a program, it is time to run a memory test. In any case, the SDOSDISKVALIDATE program should be run on all disks in use at the time of the error.

7) Power failure occurs. Re-boot and run the SDOSDISKVALIDATE program on all disks that were in drives at the time of power failure. Data files on your disks may be damaged by the power failure in a way undetectable by SDOSDISKVALIDATE, so be sure to keep your data safely backed up!

8) If an error occurs that is listed under SDOS Error codes, then it is likely that the application program has a "bug" in it; this should be reported to the applications programmer. Messages that cannot be found under SDOS Error codes are generally from the application program; refer to the documentation on the application program.

SDOSDISKVALIDATE

INTRODUCTION

The SDOSDISKVALIDATE program is used to recover from damage to the file system on a disk. This damage can occur because of a power failure, an electrical short, hardware failure or software bug, which can cause SDOS or the hardware to incorrectly update system disk data used to retrieve user data. This prevents SDOS from later being able to access the user data correctly. Use of a damaged disk without repair will usually cause more damage to the disk file structure and consequently to the user's data.

File system damage due to system crashes or bugs does not generally manifest itself right away; more often, it acts like a time bomb. To ensure that an insidious fault is not present, we recommend running SDOSDISKVALIDATE at least once a week on all disks being used heavily. It should also be run on rarely-used disks before applications are run.

The SDOSDISKVALIDATE program examines the entire disk file structure, locating and correcting any damage which might exist. Note that SDOSDISKVALIDATE cannot recover user data that is lost or damaged.

It is inappropriate to view the SDOSDISKVALIDATE program as a miracle fix-all program. Rather, the program and the user are a team with each doing what he does best: the program doing computation and searching for errors, the user providing common sense and intuition. Because both have capabilities the other lacks, the two together can do far more than either one alone can.

The program works roughly in the following manner: the program analyzes the file structure on a disk, checking for consistency. When the program finds a problem, it tells the user and either provides him with a set of options or asks him for correct data to replace the bad data. How does the user know what is right? Basically, it comes down to common sense and familiarity with the disk that is being validated. For example, if the program claims to have found an invalid file name (i.e., one that SDOS never would have allowed to be created) and said filename was "SDOSDISKVALIDˆATE" (where ˆA is control-A), a good guess is that the correct file name is "SDOSDISKVALIDATE".

Another technique is to examine a similar disk (i.e., the backup for the damaged one) to determine values for all the tuning parameters, BOOT:DIRLSN, and other data required. (This may require the user to stop validation, examine another disk with something like BMP, and re-validate the damaged disk once the critical data needed has been found.)

In some places, a high level of systems programming skill is required in order to retain a maximum amount of user data on a damaged disk (this is one of the few unfortunate side-effects of the sophistication of the SDOS file structure). Unsophisticated users can generally use the default options to repair the disk; some data files will probably be lost, but the disk will be safe to use.

RUNNING THE PROGRAM

Before running the program, one thing must be understood: the SDOSDISKVALIDATE program is designed to eliminate errors in the file structure on a disk. It doesn't care how it eliminates the error--it would just as soon delete a damaged file as fix the file. The user is responsible for guiding the program down the correct path. So if there is a critical data on the damaged disk, make a backup of the disk before beginning, just to be on the safe side.

Since there are numerous different things the SDOSDISKVALIDATE program does (so many that the program is split into 5 overlays), it would be nearly impossible to detail the inner workings of the program. Therefore, the instructions for running the program are rather simple: run the program and answer all the questions. While this seems like hardly enough instruction to run a program that is about to manipulate the whole file structure of the disk, most of the questions asked are self-explanatory (with some help from the SDOS manual) and are of the YES/NO variety. While running the program, notice that before each message and each question there is a three digit number. That number is an index into the table which follows these instructions. This number can be used to look up the description of what is happening, what the computer wants done, and what the best bet is. With this scheme, the user doesn't have to learn about terrible things like headercluster/directory conflicts until there is one.

Many of the questions asked will have a default response, which is indicated by typing simply <carriage-return>. The default responses are to designed to allow validation of a disk with minimal knowledge, and maximal safety to the user data. Note again, that due to the essentially infinite number of ways a file structure can be damaged, that this is not always true; it is just a best try.

The SDOSDISKVALIDATE program can be stopped at any step by striking the ESCape key, and the validation process continued later by re-running the program.

WHEN TO RUN THE PROGRAM

Validate all disks that were being used when the system crashes, or even when things just start acting funny. Validate any disk that gets a disk I/O error. If the SDOSDISKVALIDATE program does not give a disk a clean bill of health, run it again.

Example:

```
.sdosdiskvalidate
(101) SDOS Disk Validate and Repair, V1.1h
(102) Validate which disk (DISK: is default): d1:
(103) Device dismounted.
(104) Options (N=Noisy, V=Verify, <CR>=none)?
(115) DIRECTORY.SYS directory entry Validated.
(113) Chaining to SDOSDISKVAL.PAS4
(400) SDOSDISKVALIDATE Pass 4 V1.1h
(412) Chaining to SDOSDISKVAL.PAS5
(500) SDOSDISKVALIDATE Pass 5 V1.1h
(501) Reconstructed DiskMap matches DISKMAP.SYS contents.
```

SDOSDISKVALIDATE MESSAGES

#	MEANING

101 No meaning, just the banner.

102 At this point, the program wants to know the name of the device (i.e., disk drive) it is supposed to check out. Normal answers might be "dØ:", "fdØ:", etc. If, just after this point, the program gets an error, it is probably because a bad device name was entered.

103 This means that all sector buffers for this device have been purged and that SDOS no longer remembers what the last bad LSN was (if any). The reason this is done is so that the SDOSDISKVALIDATE program doesn't find the same bad cluster twice.

104 An option of "n", i.e. Noisy, causes the program to print out all the pertinent information about the disk which is contained in the boot file. An option of "v", Verify, does the same but also lets the user change some of the non-critical information (specifically, the disk id, the date, MINALLOC and MIDALLOC). Just hitting <cr> (no options) causes the program to proceed without printing the boot sector parameters if there are no problems.

105 The boot sector, which is critical to the use of the file structure on that disk, cannot be read by the hardware! What the program is offering to do is to write a readable, but GARBAGE boot sector. Then (hopefully), it can once again access the boot sector. Reconstruction of its contents will be necessary. Since reconstructing a boot sector requires "magic" numbers, it might be helpful to run the Validate program in noisy mode on a good disk and write down the parameters it prints out, specifically NSPC and the map algorithm, before coming back and continuing. To abort here, hit <ESC>.

106 If the boot checksum is incorrect, it means either of two things--the checksum is damaged or the boot sector is damaged (probably the latter). In any case, to get out at this point, type in "no"; otherwise tell it "yes".

107 "No"--so the program is going to stop right here. The disk is still damaged and is unusable.

108 "Yes"--so the checksum is fixed.

109 The boot file says that the disk does not have an SDOS 1.Ø file system on it--either it is incorrect or it has something like a 1.1 or 2.Ø file system on it. If it is really an an SDOS 1.Ø file system disk, type "Yes". Otherwise, type no. (Note: SDOS 1.1 uses SDOS 1.Ø file

110 The number of sectors per cluster is Ø (zero) according to the disk--if this were really true, it would have an infinite number of clusters! Therefore, NSPC is clearly wrong.

111 This is purely informative--it tells the number of sectors on the disk (NLSN), how many clusters there are (NLCN), the number of sectors per cluster (NSPC) and the map algorithm.

112 The program could not find a valid DIRECTORY.SYS. Pass 2 of the program is designed specially to ferret out hidden or broken DIRECTORYs.

113 Well, every thing here looks good--no bad LSNS, no options--so off to pass 4 to check out all of the files in the directory.

114 Either the program is in noisy/verify mode and/or there is a bad LSN. Nothing to worry about.

115 The DIRECTORY.SYS directory entry seems to be in good health, and so DIRECTORY.SYS can be OPENed by SDOS without trouble (this is a prerequisite for Pass 4).

116 A disk I/O error summary is being printed. The information supplied is identical to that given by the DISMOUNT command to the command interpreter, and is used primarily to aid in hardware problem prevention and/or diagnosis.

117 Not used.

118 A new value for Number of Sectors per Cluster is needed. The default is the value that SDOSDISKINIT would have supplied.

119 NSPC is so small that the Number of Logical Clusters for this disk exceeds the design limits of SDOS. SDOSDISKINIT would not have allowed initialization of a disk with this value, so NSPC must be wrong.

120 None of the changes to the Boot sector have actually been placed on the disk yet. Answering "Yes" will cause the boot sector to be permanently updated; "No" will leave the boot sector unchanged.

121 BOOT:DIRLSN is a Logical Sector number which tells SDOS where the DIRECTORY.SYS entry in the directory is located on this disk. The LSN so indicated is simply invalid. A guess, based on SDOSDISKINIT's default, is displayed; the user must specify a valid value to continue.

122 The DIRECTORY.SYS entry cannot be read because of a disk error or a hardware malfunction. Pass II will help solve this problem.

123 The Header Cluster of the directory cannot be read due to disk or hardware failure.

124 BOOT:DIRLSN doesn't point to a sector containing the letters 'DIRECTORY.SYS' in the first 16 bytes, as expected.

200 Pass 2 Banner. Pass 2 locates DIRECTORY.SYS on damaged disks.

201 There is a problem with the disk. The information printed here could be useful later.

202 Actually, what was printed here is just about all that can be said. Note that these faults only MIGHT be faults; if one of them is fixed, the others can change. The order of printing is from high to low in probability of fault.

203 Is the map algorithm wrong? A reasonable value for the map algorithm might be found by running this program on another disk of the same type and noticing what its map algorithm is--then trying its map algorithm (while looking at the other disk, remember what its NSPC was, too).

204 What is the map algorithm? (Read the description for message 203).

205 Is the number of sectors per cluster wrong (NSPC)? (see message 203 for related problem).

206 If you know what the map algorithm should be, say "Yes". Otherwise, SDOSDISKVALIDATE will try to find a map algorithm that works (holding BOOT:DIRLSN and NSPC constant). This process takes several minutes. Values displayed may be entered as new map algorithm values; if BOOT:DIRLSN or NSPC are wrong, the values displayed are probably incorrect. If no values are displayed, DIRLSN is probably incorrect, or the DIRECTORY.SYS entry is damaged beyond recognition.

207 The header cluster specified by the DIRECTORY.SYS directory entry does not appear to be a header cluster. Either it is destroyed, or the header cluster number specified by the DIRECTORY.SYS entry is incorrect. A list of possible faults is given.

208 The directory entry for DIRECTORY.SYS points to a header cluster, but the header doesn't look like a header (i.e., it doesn't point to itself like it should). A different header cluster can be selected here; the value generally selected by SDOSDJSKINIT will be displayed if the user responds "Yes".

209 There's something out there where "DIRECTORY.SYS" should be but it isn't right. It can be changed to say the proper thing, but this is generally not the cure; try changing NSPC or MAPALGORITHM first.SYS" if desired.

210 "Yes" causes the header cluster to be fixed. "No" (generally the right answer) means something else will have to be tried.

211 "No"? There are no other options to try. The disk structure is left in a damaged state.

212 Perhaps the disk or hardware is unreliable, and re-reading will succeed in fetching the sector. Re-trying is generally worth the effort.

213 Since directory entry cannot be read, a garbage sector could be written in its place, and reconstruction attempted. Unless BOOT:DIRLSN or the map algorithm are wrong, this is really your only option.

214 Changes made by the user are being made permanently by writing them back to the disk.

215 Enter the new number of sectors per cluster (NSPC) (see message 205).

216 Perhaps the problem is that BOOT:DIRLSN points to the wrong sector. This provides the user an opportunity to specify a new hexadecimal value for BOOT:DIRLSN. The default value used by SDOSDISKINIT is shown.

217 The user must enter a new DIRLSN here as a hex number. If the user knows the exact value, he can say "yes" and save some time. If not, SDOSDISKVALIDATE will scan the entire disk (using the current map algorithm) looking for sectors which appear to contain the DIRECTORY.SYS directory entry. Values printed out may be entered as a new BOOT:DIRLSN; since they depend on the map algorithm, if the map algorithm is wrong, these values will probably be incorrect.

218 Since some things dealing with the boot sector were changed, should the program update the boot sector to reflect the changes (otherwise the changes are not permanent)? Say "yes" unless there is a reason not to.

219 This much is okay, so on to check out the non-critical portion of the file system.

220 The header for the directory is alleged to be at an illegal cluster. A list of probable faults is given.

221 The letters "DIRECTORY.SYS " were found where BOOT:DIRLSN indicated; so (apparently) the directory entry has been located. This does NOT mean the directory entry is the right one (coincidence?) or that it is rational. A thorough check on the directory entry is now performed.

222 The DIRECTORY.SYS file appears to be OPENable, so other disk validation steps can proceed.

223 DIR:HCSIC is almost always 1; under rare circumstances, it can be 2. Other values are almost certainly wrong.

224 A Header Cluster for DIRECTORY.SYS must have at least two clusters: one for the header, and one for a data cluster to hold directory entries. A list of possible faults is given.

225 The number of data clusters in DIRECTORY.SYS is more than the number of clusters on the disk; this is obviously incorrect. A list of possible faults is given.

230 The DIRECTORY.SYS directory entry has an impossibly large file size. A list of possible faults is given.

231 The DIRECTORY.SYS directory entry gives a filesize which is not a multiple of the cluster size, which it must be. A list of possible faults is given.

232 Answer "Yes" if the size of DIRECTORY.SYS is wrong. A reasonable value (based on number of LCNs) is displayed.

233 Enter a number that is a multiple of the cluster size (in bytes).

234 There are gaps (dummy LCNs) in the Header Cluster of DIRECTORY.SYS. The header cluster is damaged, or NLCNs given by the directory entry is too large. Other possible faults are also listed.

235 A "Yes" answer allows a gap to be filled with a specified cluster number. SDOSDISKVALIDATE will suggest a cluster number to be used, based upon the "previous" LCN in the header cluster. Do not take this value too seriously.

236 Enter cluster number to be used to plug the gap.

237 A "Yes" answer is required if the number of LCNs for DIRECTORY.SYS needs to be changed. A good guess as to its proper values has already been displayed as the value according to the header cluster (at least two LCNs are required for a valid directory.)

238 Enter the exact number of clusters owned by DIRECTORY.SYS.

239 What appears to be the DIRECTORY.SYS is not, or it is damaged, as it is marked as non-existent.

240 This gets around an initial problem so that further checking can be performed. A no answer will require changes to BOOT:DIRLSN or the map algorithm to locate the true DIRECTORY.SYS.

241 The DIRECTORY.SYS header cluster does not show BOOT:DIRLSN as part of the DIRECTORY.SYS file, which it must be. Probably the header cluster is damaged. A list of possible fauls is given.

242 The cluster number containing BOOT:DIRLSN will be added to the header cluster if a "Yes" response is given.

243 The header cluster for DIRECTORY.SYS is filled to a sector boundary (this is extremely unusual). HCSIC will have to be increased before BOOT:DIRLSN can be added to the header cluster.

244 The number of data clusters owned by the header cluster does not match the count specified by the DIRECTORY.SYS directory entry. A list of possible faults is given.

245 HCSIC specified by the DIRECTORY.SYS entry is not legal if NSPC is correct.

300 Pass 3 Banner. Pass 3 handles BADCLUSTERS.SYS, and allows modifications to non-critical file system parameters.

301 This is the disk ID (the name of the disk).

302 If the name of the disk is inappropriate or garbage, enter an appropriate name.

303 Double quotes aren't allowed. Leave them out.

304 MINALLOC and MIDALLOC are the minimum file allocation and the minimum file extent, respectively. If these numbers are bigger than 10 or so, they're probably wrong.

305 Say "Yes" if you wish to change either one.

306 Try something in the range 1 to 10.

307 Use a value about double that of MINALLOC.

308 Date of creation (SDOSDISKINIT) of the disk.

309 If creation date is wrong, enter the new one in the form MM/DD/YY.

310 Since some of the data in the boot sector is modified, the program wants to rewrite the updated sector on the disk. (Yes is the normal response).

311 Recently, the system got either a hardware read or write error on a sector, and SDOSDISKVALIDATE is trying to hide it away in a pit for bad sectors so it won't get used again. This pit, called "BADCLUSTERS.SYS", hasn't been created on that disk yet, so SDOSDISKVALIDATE can't hide the bad sector. This means that the bad sector will be back to plague the system later. Finish validating and create the file BADCLUSTERS.SYS on your disk. Eventually the bad sector will reappear, and this procedure will then be successful.

312 Either BADCLUSTERS.SYS is damaged or the disk has an enormous number of bad sectors. If BADCLUSTERS.SYS is damaged, SDOSDISKVALIDATE will fix it if allowed to run to completion. If the disk really has a large number of bad clusters, we recommend replacing the disk.

313 Apparently the ghost of the bad sector still haunts us--not much to do about it now. The problem should not re-occur.

314 The sector which caused a read or write problem is now tucked away in a harmless place. However, it is probable that some file owns the bad sector, in which case that file and BADCLUSTERS.SYS now overlap because now both files own the cluster containing the bad sector. Pass 4 will help resolve the overlap.

315 Done with Pass 3.

316 This is the logical sector number on which the last unrecoverable read/write error occurred (note that the logical sector number may not be the same as the physical sector number).

317 Disk IDs have a limited length. Use a shorter ID.

318 Program is making the changes permanent.

400 Pass 4 Banner. Pass 4 checks the consistency of all files in the directory.

401 This file name isn't legal--SDOS would never have created it so it must be damaged.

402 Supply a new name for this file (try Bad1 then Bad2...) or the program will just skip over this file altogether.

403 The directory entry for this file is damaged. The validate program has presented the nature of the damage and is asking for instructions on what to do about it. If the file has no value to the user, pressing ESCape will abort the request and provide the user an opportunity to simply delete the file.

404 There was a disk I/O error in trying to read the header for this file. If read again, the error may go away.

405 This file's header doesn't point to itself--so either the header is damaged or the directory entry is pointing to the wrong place.

406 Enter <CR> if you don't know.

407 It is unlikely that the user knows. Inspection of a backup disk might reveal the value.

408 If lost, just type in the number printed out above (colon and all). Otherwise, type in the number that is right.

409 The program can fix the header, leave it in its damaged state, or delete the file--type in "f", "l" or "d".

410 The file owns a cluster that doesn't exist.

411 The directory entry cluster count is inconsistent with the actual number of clusters allocated.

412 Done here, now to check out DISKMAP.SYS.

413 These two files overlap each other at this LCN. The most likely cause of this is that one file (which was created last and thus has the latest date in the message) is intact while the other file has been overwritten in places. The best course of action is to deallocate the cluster from the older file, since the data in that cluster probably belongs to the younger file. If a file overlaps BADCLUSTERS.SYS, then the cluster should be deallocated from the file, as it has a sector in it which causes read/write faults.

414 From which file should the cluster be removed? (If nothing else, say "neither"--at least that way nothing happens).

415 SDOSDISKVALIDATE won't do it unless you are absolutely sure. Gone is gone forever.

416 SDOSDISKVALIDATE won't do it unless you are absolutely sure. Gone is gone forever.

417 DIR:HLCN has an illegal value.

418 What is the header cluster number? If unknown, just hit return.

419 One doesn't fix an invalid cluster number by substituting another invalid cluster number.

420 DIR:HCSIC is the number of sectors in the header cluster which contain valid system data. It must be smaller than BOOT:NSPC.

421 A good number here is (LCNS*2)/NBPS rounded up. For small files, this is generally 1. For big files, try 2 or 3. Really big files require BOOT:NSPC here.

422 Type in a number from 1 to NSPC.

423 With the parameters specified in the boot sector, SDOS can't keep track of that many bytes in a file!

424 That's the most SDOS can keep track of.

425 The file protection byte has been garbaged.

426 Enter any combination of "B" or "W" for Backup or Write protect. Enter <CR> if the file does not need protection.

427 "Yes" is the best answer. "No" generally means you get to say yes, over and over, later.

428 "Yes" makes this particular bad cluster pointer go away permanently.

429 Yes, no, or <CR> only, please.

430 Since there have been problems with this file, it might be best to delete it.

431 Even though there are changes, the disk hasn't been updated to reflect these changes yet.

432 Even though SDOS gave us a read error, it might go away on another try.

433 The date given in the directory entry is bad.

434 If the directory sector is really unreadable, then DIRECTORY.SYS will probably be unusable (because SDOS reads it sequentially). Writing on it might make the directory usable. Some files will probably be lost, but they are unaccessable anyway.

435 Two files in the directory own the same header cluster (this is like having two seperate chapters in a book, both starting on the same page). One of the files must be deleted. Generally, deleting the oldest is appropriate.

436 The file's header owns the same data cluster twice (this is like a book having two of the same page number). One of double-ownerships is wrong. The values printed describe the doubly owned cluster, and displacements into the header cluster where they were found. One of the duplicates must be removed.

437 A sector of DIRECTORY.SYS can't be read. Try reading it again once or twice; if it can't be read, some files will probably be lost.

500 Pass 5 Banner. Pass 5 checks DISKMAP.SYS, and verifies that BOOT.SYS and SDOS.SYS are properly constructed.

501 This is normal.

502 "Yes" is the best answer. "No" is painful. There are some clusters that are marked as owned, but aren't owned by any file.

503 Freed cluster numbers are shown.

504 Type "yes" unless there is a specific reason not to.

505 It is marked as free but this file owns it...so SDOSDISKVALIDATE will mark it as being allocated.

506 Why keep them allocated?

507 This is how much disk space is reclaimed.

508 All the disk map problems are fixed.

509 Say "yes"...it should be preserved in its ruined state.

510 SDOS blew it!....but SDOSDISKVALIDATE will try again (and again...until SDOS gets it). This is a good indication that SDOS is about to die (or the hardware is ill).

511 The disk map has been updated properly.

512 DISKMAP.SYS is left in its damaged state. Extend files or create new files on this disk at own risk, a disaster is almost sure to result.

513 Can't find DISKMAP.SYS; there's nothing else that can be done.

514 Two types of damage can occur:
1) a cluster is marked as allocated in the diskmap when it shouldn't be, i.e. It is not part of a file. This condition only makes SDOS run out of disk space faster.
2) a cluster is not allocated when it is part of a file. If left unfixed, this could lead to overlapping files which results in lost data.

515 There is at least one cluster which should be allocated but isn't. SDOSDISKVALIDATE can print out the file name associated with every cluster that suffers this problem. This can indicate which files might have been accidentally overlapped by others, and thus have damaged content. However, the amount of time the program takes to find out which file is associated to a cluster varies from short to very long. Thus, it may be desirable to skip this to speed up the time it takes to fix the disk.

516 DISKMAP.SYS is improperly constructed; NCLNs is <2 or an invalid data cluster is allocated for the first data cluster. SDOSDISKVALIDATE can choose an unused cluster and fix the problem. "YES" is about the only choice you have.

517 OPENed DISKMAP.SYS, but cannot find it in DIRECTORY.SYS. Something is very sick.

518 DISKMAP.SYS file size has wrong value. A reasonable reply is "YES".

519 Report this problem to Software Dynamics.

520 Only one DISKMAP.SYS entry is allowed. It probably is safe to delete the duplicate.

521 There is a duplicate DIRECTORY.SYS entry. Only one is allowed. The duplicate is probably the right one to delete.

522 There is a duplicate BOOT.SYS directory entry; only one is allowed. It is probably safe to delete this one.

523 BOOT.SYS must have a certain minimum size to hold a Disk ID.

524 BOOT.SYS must own cluster number Ø.

525 Not used.

526 SDOS.SYS must own cluster number 1.

527 Only one SDOS.SYS directory entry is allowed. It is probably safe to delete this duplicate.

528 Not used.

529 SDOS.SYS must be a contiguous file. Answering YES will make it contiguous, but will also destroy its contents. It might be wise to save its contents before proceeding.

530 Fixing a problem in this pass caused the contents of DISKMAP.SYS to become incorrect. Pass 4 will re-correct DISKMAP.SYS contents.

531 This is the number of errors in the real diskmap. If it is small, displaying all the errors might be informative about damage to files. If large, the display is generally long and tedious because the diskmap was probably completely destroyed; many data files might be damaged. "D" says show all files that might be damaged; "F" says "fix the map, I don't care", and "Q" says "leave the map damaged". Only experts use the Q option.

532 Not used.

533 A BOOT.SYS file must exist.

534 Cannot proceed without BOOT.SYS.

DESCRIPTION OF SDOSDISKVALIDATE PASSES:

This section can be skipped by all users not interested in the structure of the validation program.

Pass One (SDOSDISKVALIDATE):

Pass One is specifically designed to check the boot sector. This pass will make sure that the Directory can be opened by SDOS.

Problems	Action Taken
No problems,	
no options	Chain to Pass 4
options	Chain to Pass 3
Boot sector gets	
read errors	Write garbage boot sector out, try to fix up
Boot gets checksum	
error	Either fix it or quit.
Boot has wrong	
file version	Fix or quit.
Boot claims NSPC=Ø	Complain, chain to Pass 2.
Boot looks ok,	
but can't find	
directory	Chain to Pass 2.

Pass Two (SDOSDISKVALIDATE.PAS2):

Pass Two attempts to fix up the disk file structure well enough so that SDOS can open the disk's DIRECTORY.SYS file. To determine if the directory is really OPENable, Pass Two uses this criteria: The directory entry must contain a pointer to a header such that directory's header 1) points to itself and 2) points back to the data cluster. Also, the entry for the directory must contain DIRECTORY.SYS as its file name. If this criteria is met, SDOS can open the directory.

Otherwise, Pass Two allows the user to juggle the NSPC and Map Algorithm until these criteria are met, or failing this, allows the user to construct a directory with the hope that only one little thing is damaged and fixing that problem will allow SDOS to open the directory. If the directory is not OPENable, all files on the disk are lost permanently.

If this doesn't succeed, the validate will probably have little effect on your disk. At this point, either give up or call your retailer/distributer/SD for help.

Pass Three (SDOSDISKVALIDATE.PAS3):

Pass Three does two things:

Allow the user to examine/modify the non-critical data in the boot sector,
and
Take care of any sector which has been causing disk read/write errors.

The data Pass Three allows the user to see/change:

The Disk ID
The Minalloc
The Midalloc
The Creation Date

The bad sector (if any) is merely appended to the end of BADCLUSTERS.SYS (unless it is already present in the file). Come Pass Four, the user will be notified as to which file it overlaps.

Pass Four (SDOSDISKVALIDATE.PAS4):

Of all the passes, Pass 4 is the most complex. Pass Four is responsible for checking the Directory entries and headers for all files on the system.

Pass Four cycles through the entire directory, performing the following checks on each file:

1) Is its file name valid?
2) Is the data in the entry reasonable?
3) Does the header cluster contain a pointer to itself?
4) For each entry in the header:
   a) Is it legal?
   b) Does it not overlap another file?
5) Does the number of clusters in the header match how many the directory says it should have?

If the answer to all these questions is yes, then the file is declared good. If not, the program allows the user to make any changes which will correct the data.

At the termination of the the pass, Pass Four chains to Pass Five.

Pass Five (SDOSDISKVALIDATE.PAS5):

Pass Five checks the DISKMAP.SYS and corrects any inconsistencies that it finds, specifically a cluster not being allocated when it should or being allocated when it shouldn't.

It checks that BOOT.SYS exists, and has the proper structure. It also verifies that SDOS.SYS is contiguous if it exists.

SDOSSET

SDOSSET is used to describe CRTs and printers to SDOS. A portion of SDOS known as the "Virtual Terminal Driver" (VT driver) is responsible for handling all CRTs and/or printers attached to an SDOS system (real CRTs and printers are thus called "VT devices"). For technological, historical and competitive reasons, most CRTs and printers are controlled in different ways, even to accomplish the same effect (consider the differences in the designs of automobile and truck engines from different manufacturers). It is the purpose of the VT driver to make these devices look as alike as possible (all cars and trucks have gas pedals and steering wheels in virtually identical places), so screen-oriented editors or applications need not be aware of every possible peripheral to operate with them all correctly (once you've seen one gas pedal and steering wheel, you've seen them all). This is accomplished by 1) defining an SDOS-standard "Virtual Terminal" with a reasonable compromise of capabilities available from all devices, 2) requiring application programs to issue input, output, positioning and other requests in a form compatible with the Virtual Terminal definition, and 3) having the VT driver convert such standard Virtual Terminal I/O operations into the actual operations required by the physical CRT or printer used.

The SDOSSET program is used to describe certain characteristics of VT devices to the Virtual Terminal driver, so the VT driver can correctly implement the standard Virtual Terminal operations on those devices. The VT driver needs to know page width and depth, how to clear the screen, position the cursor, and other sundry details. The SDOSSET program can also set some parameters which are invisible from the point of view of the application, such as Tabs, Baudrate and how long a device timeout takes.

Information about how to control a particular CRT or printer is generally packaged in a form known as a "profile". Individual profiles correspond to specific models of a particular manufacturer's terminal or printer. SDOS systems generally come with several "standard" profiles (those deemed interesting by the computer manufacturer), and a few "malleable" (changeable) profiles, which can be "modified" to describe a terminal or printer which is not covered by the standard profiles. SDOSSET can be made to tell a user which profiles the system knows about, and can be used to specify to SDOS that a particular standard or malleable profile should be used for a particular device, instead of the one being currently used (when SDOS is first booted, default profiles defined by the manufacturer are automatically assumed for all VT devices attached to the system; these can be automatically overridden by placing SDOSSET commands in the file INITIALIZE.DO).

A standard profile specifies, for a particular model and make of device, the (default) page Width, (default) page Depth, (default) Timeout, how to perform a New Line, how to perform Clear Screen (CRT)/New Page (printer), how to perform Erase-To-End-of-Line, whether the device should Wrap (break lines which are too long and continue on the next line) or not (simply chop excess from lines which are too long), Coloring (how to make text appear differently without changing its meaning [bold, blink, etc.]) and Input Translation (how to map input keystrokes into SDOS-standard VT keys). SDOSSET can be used to specify use of a standard profile for a particular device, and optionally override the default values.

A "malleable" profile specifies trivial defaults for all the details a standard profile would supply. SDOSSET can be used to specify a malleable profile if no standard profile seems to match the actual device, and then can be used to adjust these details (except Coloring and Input Translation) to match the actual device. Details for the required adjustments are generally easily obtained from the CRT/printer's User Documentation.

To properly set up a VT device using a malleable profile, the sequences of special characters to accomplish Cursor Positioning, Clear Screen/New Page, moving the cursor to a new line, and Erase-To-End-of-Line must be given. Sometimes such sequences require a certain amount of time to take effect on the device; this is accomplished by specifying, for each sequence, how many (maximum 255) Idle (Ascii:null) characters must be sent after the sequence to accomplish the desired delay. For devices which cannot accomplish these functions on their own, the VT driver can be made to simulate the desired effect. Sometimes, the simulated effect is more useful than the standard profile for a device (see CLEARSEQ description, below), and so a malleable profile will be selected in spite of the presence of a matching standard profile for a device.

For some devices, it may not be possible to specify a sequence to accomplish the desired effect, nor will the simulation supplied by the VT driver be suitable. Such a device cannot be handled by the VT driver properly until a "standard" profile is created for the device, and configured into the system. Refer to VTDRIVER section of "SDOS Systems Implementer's Guide" for more detail.

SDOSSET can be invoked as a terse (one-line) command, or as a dialog. To understand the dialog, one must first be familiar with the terse form, as the dialog simply collects the information the terse form gives in a different format.

The terse form of SDOSSET is as follows:

```
.SDOSSET <device>,<parameterdefinition>,...
```

If the last character of terse SDOSSET command line is a semicolon (";"), then the semicolon is discarded, and the console is read for additional input, which is appended to the previous line. <device> must be the name of a Virtual Terminal device, or an error will be given and SDOSSET will abort. <parameterdefinition>s are defined by the table that follows. More information on these parameters can be found in the section on the Virtual Terminal Driver in the SDOS Application Programmer's Guide. Values to be used for the parameters can be found in documents provided by the manufacturer of the device. Specification of an illegal parameter value will result in an error, and that parameter's value will not be affected.

WIDTH=<number>
Specifies Page Width (default specified by Profile)

DEPTH=<number>
Specifies Page Depth (Ø=Hard Copy, default specified by profile)

WRAP
Specifies that too-longlines are to be broken and wrapped (default specified by profile)

NOWRAP
Specified that too-ling lines are to be truncated at page width

TABS=(<number>,...)
Specifies up to 16 tab stops, in ascending order. The value "one" means the left margin. The default values are every 8th column (1,9,17,...) up to 132.

TIMEOUT=<number>
Specifies output timeout in seconds. The default is specified by the profile.

BAUDRATE=<number>
Specifies baud rate for device (Ø to 65535) Whether this value is changeable is determined by the computer manufacturer. The default value is specified by the manufacturer of the computer.

PROFILE=<numberorname>
Specifies desired profile. A profile number or a profile name may be given. The default value is specified by the manufacturer of the computer. A list of valid profiles for the current computer configuration can be obtained by using SDOSSET in a dialog mode. Malleable profiles are usually named MALxxx.

NLSEQ=(CR,LF,<idles>)
Specifies CR,LF (the other alternative is LF,CR), and <idles> specifies the number of idles to follow. This parameter may only be used if the profile selected is malleable. This is the default setting.

NLSEQ=(LF,CR,<idles>)
Specifies LF,CR as the new line sequence, and number of idles to follow. This parameter may only be used if the profile selected is malleable.

CLEARSEQ=(<idles>,<c>,...)

Specifies how to clear the screen. <idles> specifies the number of idles that follow. <c> are numbers (hex or decimal) that represent the sequence of character codes, from left to right, that will clear the screen (for a CRT) or move to the top of the next page (for a printer). A maximum of 4 characters are allowed. If the sequence is empty, the VT driver will simulate a Form-Feed by outputting enough Ascii:LF (line-feeds) to reach the currently selected page depth (this is very useful on printers which have varying sizes of paper placed in them). This parameter may only be used if the profile selected is malleable.

EEOLSEQ=(<idles>,<c>,...)

Specifies how to erase to end of line. <idles> specifies the number of idles that follow. <c> are numbers (hex or decimal) that represent a sequence of character codes, from left to right, that will blank the portion of the line to the right of the cursor. A maximum of 4 characters are allowed. If no characters are given (the default), then the VT driver will simulate EEOL by outputting enough spaces to reach the right margin, and then backspacing. This parameter may only be used if the profile selected is malleable.

POSNSEQ=(<idles>,<rowdisp>,<coldisp>,<c>,...)

Specifies how to position the cursor (this is generally unnecessary for printers). <idles> specifies the number of idles that follow. <c> are numbers (hex or decimal) that represent a sequence of character codes, from left to right, that will position the cursor in the uppermost, left-hand corner of the screen. A maximum of 4 characters are allowed. <rowdisp> and <coldisp> are numbers that specify the displacement (counting the left-most as zero) into the sequence for the characters representing the row number and the column number, respectively. When positioning, the VT driver adds the desired row and column numbers to the specified character codes before outputting the sequence. The default is "no characters", which will cause the sequence "@@" to be printed when cursor positioning is attempted. This parameter may only be used if the profile selected is malleable.

The following table lists the names of profiles defined as of the time this manual was printed. New profiles will come into existence as needed, and old ones will fall into disuse. Not all of the profiles are available on every system, although MALVT and MALLPT are almost always available.

Table of Profiles

Profile Number	Profile Name	Make and Model Corresponding
1	MalVT	Changeable console (malleable)
2	ADM1	Lear Siegler ADM1
3	ADM3	Lear Siegler ADM3
4	SOROC12Ø	SOROC Technology 12Ø
5	H19	Heathkit/Zenith H/Z-19 (VT52 compatible)
6	HardCopyVT	Standard HardCopy Terminal
7	TVI912c	Televideo 912c
8	ExorTerm155	Motorola Exorcisor Terminal 155
9	Mallpt	Changeable lineprinter (malleable)
1Ø	CenLpt	Centronics lineprinter
11	rs232Lpt	Standard RS232 lineprinter
12	TI81Ø	Texas Instruments 81Ø lineprinter
13	CoCoLowRes	Color Computer low resolution screen
14	Epsonmx8Ø	Epson MX8Ø
15	Epsonmx8Øwide	Epson MX8Ø in compressed print
16	Epsonmx1ØØ	Wide Epson printer
17	VT1ØØ	Digital Equipment VT1ØØ
18	CoCo	Color Computer high resolution screen
19	HazeSP	Hazeltine Esprit

Examples:

```
.SDOSSET CONSOLE:,PROFILE=SOROC120
```

This defines the terminal as a SOROC (manufactured) IQ120 (model). No other parameters are needed, as the profile specifies all the necessary details.

```
.SDOSSET LPT:,PROFILE=EPSONMX80,TIMEOUT=25,BAUDRATE=1200
```

This specifies the LPT: device to be an EPSON MX-80 printer. Revision of the timeout value may be necessary because the profile believes the timeout should be one value, and buffer options on the printer may require the timeout to be much longer. Baudrate selection is required when a peripheral's speed is different than the current configuration of the computer hardware.

```
.SDOSSET LPT:,PROFILE=MALLPT,DEPTH=51,WIDTH=105
```

This is used to change the logical depth of the paper on the printer. After doing this, SDOS will automatically send a form character to the printer if it does not think it is at the top of a page; this ensures that both SDOS and the printer agree that the printer is at the top of a page. The user should immediately check that the printer is correctly positioned.

```
.SDOSSET CONSOLE:,PROFILE=MALVT,WIDTH=72,DEPTH=20;
>POSNSEQ=(0,2,3,:1b,:3d,:20,:20),CLEARSEQ=(10,:0c)
```

This example sets up a CRT for which no profile configured into the system is suitable. It defines a new display width and depth, as well as a clear screen and positioning sequences. The clear screen sequence consists of a single character (:0c) followed by 10 idles; the positioning sequence consists of four characters (:1b,:3d,:20,:20), two of which will be added to the row and column numbers supplied by the user of the position operation of the control syscall. The 2 and 3 are the positions within the sequence (with 0 being the first position) of the row and column addresses, respectively: the values there in the sequence will be added to the values supplied by the user. The positioning sequence will be followed by zero idles. Since the EEOLSEQ parameter was not specified, it will default to simulation.

SDOSSET can also be invoked as a dialog by typing:

```
.SDOSSET
```

The program will ask for the name of the device, and will show the user its current profile. Questions regarding modifications to the profile parameters will be asked; the user may explicitly specify a new value, or enter just <CR> to indicate "No change". Typing '?' in response to a "New Profile?" question will cause SDOSSET to show a list of profile names it understands ON THIS SYSTEM.

```
.sdosset
Set Terminal Options V1.1c
Device name? console:
Device Type = Console
Current profile = malvt
New Profile (enter name, number or <CR>)? SOROC12Ø
Current Tabs = 8,16,24,32,4Ø,48,56,64,72,8Ø,88,96,1Ø4,112,12Ø,128
New Tab Stops (enter up to 16 numbers or <CR>)?
Current Idles count = Ø
Idles to follow new line (enter number or <CR>)?
Current Width = 79
New Width (enter number or <CR>)?
Current Depth = 24
New Depth (enter number or <CR>)?
Wrap Set
Wrap at end of line (yes/no/<CR>)?
Current Baud rate = Ø
New Baud rate (enter number or <CR>)?
Current Output Timeout (in seconds) = 6.23333333
New Timeout value (enter number or <CR>)?
.
```

The example on the following page shows a user configuring a terminal for a device which is not standard on a system.

```
.sdosset
Set Terminal Options V1.1c
Device name? console:
Device Type = Console
Current profile = malvt
New Profile (enter name, number or <CR>)? Soroc120
Unimplemented profile name -- available ones are:

   1   malvt
   2   adm1
   3   adm3
   5   h19
   6   hardcopyvt
   7   tvi912c
   9   mallpt
  10   cenlpt
  11   rs232lpt
Current profile = malvt
New Profile (enter name, number or <CR>)? malvt
Current Tabs = 8,16,24,32,40,48,56,64,72,80,88,96,104,112,120,128
New Tab Stops (enter up to 16 numbers or <CR>)?
Current Idles count =  0
Idles to follow new line (enter number or <CR>)?
Current Erase to EOL sequence = NONE
New EEOL sequence (up to 4 numbers or <CR>)? :1b,:54
Current Idles following =  0
Idles to follow sequence (enter number or <CR>)?
Current Clear Screen sequence = NONE
New Clear Screen Sequence (up to 4 numbers or <CR>)? :1b,:2a
Current Idles to follow =  0
Idles to follow (enter number or <CR>)? 2
Current Position Cursor sequence = NONE
New Position Cursor sequence (up to 4 numbers or <CR>)? :1b,:3d,:20,:20
Current Row displacement within sequence =  0
Row displacement within sequence? 2
Current Column displacement within sequence =  0
Column displacement within sequence? 3
Current Idles to follow =  0
Idles to follow sequence (enter number or <CR>)?
Current Width =  80
New Width (enter number or <CR>)? 79
Current Depth =  24
New Depth (enter number or <CR>)? 24
Wrap Set
Wrap at end of line (yes/no/<CR>)? yes
Current Baud rate =  0
New Baud rate (enter number or <CR>)?
Current Output Timeout (in seconds) =  6.23333333
New Timeout value (enter number or <CR>)?
```

The SDOSERRORMAINT Program

The SDOSERRORMAINT program is used to manipulate the ERRORMSGS.SYS file. Commands are provided for examining, changing, deleting, and adding error messages. This program will create an ERRORMSGS.SYS file if one doesn't already exist.

There is a special DO file called ERRORMSGBUILD.DO which will create an ERRORMSGS.SYS file and insert all the standard system error messages. To use this standard procedure, simply type:

```
.DO ERRORMSGBUILD.DO
```

on the console. This procedure uses the SDOSERRORMAINT program to actually build the file.

For non-standard messages, or for changing/adding/deleting messages in the file, the SDOSERRORMAINT program is used. The commands for this program are given below.

C	Create new ERRORMSGS.SYS file
V	View (examine) a message
I	Insert a message (replaces old message)
L	List messages, from message number to message number, on console
D	Delete a message
O	Output messages, from message number to message number, on file/device
S	Stop, exit to command interpreter

Each command is typed as a single letter followed by a carriage return when the program presents its prompt '>'. The program will prompt with questions when additional parameters are needed.

To create a hardcopy printout of all the errors currently defined for an SDOS system, the following procedure can be used:

```
.SDOSERRORMAINT
Errormsgs.sys maintainence program V1.Ø 4/25/8Ø
>o
OUTPUT FILE: LPT:
FROM: Ø
TO: 65535
>s
.
```

STANDARD SDOS ERROR CODES

Ø - Program completed normally
1 - Operator requested Attention
2 - Value Stack Overflow (expression too complex)
3 - For-Next Stack Overflow (too many active FOR-NEXT loops)
4 - NEXT without FOR
5 - Gosub Stack Overflow
6 - RETURN without GOSUB
7 - Conversion Error
8 - Input Buffer Overflow
9 - Array or Vector Subscript out of range
1Ø - Runtime package self-checksum failed-->Suspect damaged RTP or bad memory
11 - String Subscript out of range
12 - String subscript too large
13 - Undefined Line Number encountered
14 - Arithmetic Overflow
15 - Non-Integer operand to Logical operator (& ! XOR COM ** HEX$)
16 - Concatenated String exceeds CATMAX
17 - Tab count > 255
18 - Invalid FORMAT string
19 - I can't store that value into a byte
2Ø - Illegal Argument to SIN/COS/TAN/ATN
21 - Logarithm of Ø or negative number
22 - Square root attempted on negative number
23 - PEEK or POKE address < Ø or > 65535, or not an integer
24 - POKE value < Ø or > 255, or not an integer
25 - Attempt to POKE runtime package
26 - Version number doesn't match BASIC Runtime Package
27 - Wrong number of arguments to function/subroutine
28 - Data space for BASIC program overlaps SDOS
29 - Basic Program overlaps Runtime Package
5Ø - Channel number > 255
52 - File name is too long
6Ø - File position < Ø or >= 2^31
1ØØ - Compilation or Assembly had fatal errors
1Ø1 - Warning errors issued by Compiler or Assembler
1Ø2 - Bad Command Format
1Ø3 - Can't do GOTO from CONSOLE:
1Ø4 - Program terminated abnormally
1Ø5 - Insufficient memory to execute command
2ØØ - Syntax Error
2Ø1 - Can't find branch target
2Ø2 - Can't find "]"
2Ø3 - Can't Branch into Bracket Pair
2Ø4 - *** EDITor error ***
2Ø5 - Illegal argument for command
2Ø6 - Zero is not a valid argument
2Ø7 - Command requires argument
2Ø8 - Command doesn't want an argument
2Ø9 - No such "E" command
21Ø - Illegal character

211 - Can't use that as delimiter character
212 - Too many)s
213 - Too many (s
214 - Xchange not valid, must do search or insert first
215 - Command not allowed while doing edit with EB
216 - Can't find string
217 - Q register index must be 1 to 9
218 - Need to open input file first
219 - Text Buffer is full
220 - Command buffer is full
221 - Don't have enough lines in buffer to J that far
222 - Illegal tab stop list
223 - Need to select output file first
224 - Unbalanced [ ]s
225 - Bracket Stack overflow or underflow
226 - End of File prior to "A" or "EY"
227 - Buffer approaching full, operation aborted
228 - Error encountered during EDIT for which no recovery was provided
229 - Overflow occurred in operation
230 - Can't find matching [
1000 - BOOT sector checksum failed
1001 - End of File encountered
1002 - Can't DISMOUNT because a file is OPEN
1003 - No Debugger present
1004 - Bad File Position Requested
1005 - Number of Bytes per Cluster is too big (> 65535)
1006 - No DISKMAP.SYS file, can't allocate or free disk space
1007 - No matching File Control Block found
1008 - No DEFAULTPROGRAM on default disk
1009 - File is Delete protected
1010 - File is Write protected
1011 - No such file
1012 - Logical Cluster Number out of range
1013 - Length of file name > 16 characters
1014 - New file already exists
1015 - Disk space exhausted
1016 - I tried to free an unallocated cluster
1017 - No more free FCBs (***SYSTEM***)
1018 - File system is incompatible with current file system (Version 1.0)
1019 - File is being CREATEd
1020 - Disk is mounted, can't change Map Algorithm
1021 - Renamed-to filename isn't legal
1022 - No ERRORMSGS.SYS file on drive 0 (think about this!)
1023 - File name doesn't start with A thru Z or $
1024 - Illegal file Size specification
1025 - Header cluster not initialized for RDCN fetch (***SYSTEM***)
1026 - Not enough (CNFG:)DSKBUFFERPOOL in I/O package
1027 - Disk Driver doesn't implement power fail
1028 - Can't load that, not load format file
1029 - Bad file version number

1030 - Channel number is too big
1031 - Channel is already open
1032 - Channel is closed
1033 - Illegal SYSCALL number
1034 - Illegal Device operation requested
1035 - Can't rename to a different device
1036 - SDOS load record format error
1037 - Program too big to load
1038 - Illegal LSN passed to physical disk drivers
1039 - DIRECTORY.SYS is damaged!
1040 - Input buffer overflow in driver
1041 - *** Program killed ***
1042 - Device timed out
1043 - Sector size is not a power of 2! (I/O package fault)
1044 - *** not used ***
1045 - Disk read error
1046 - Disk write error
1047 - Disk seek error
1048 - Disk is write protected
1049 - Disk device is software write locked
1050 - SDOS self test checksum error!
1051 - Number of LSN's >= 2^24, I quit! (I/O package error)
1052 - Cluster size is too small to support a file that big
1053 - SYSCALL block is too short
1054 - SYSCALL Read-back buffer is too short for reply
1055 - SYSCALL Write data buffer is too short
1056 - No such device in this configuration
1057 - Device errored
1058 - Device must be a disk
1059 - Channel 0 is not open to the CONSOLE:
1060 - Device not ready
1061 - TIME not set
1062 - No such logical unit number
1063 - No start address supplied
1064 - No such program
1065 - Old file by same name already exists
1067 - Disk space allocator called with request for 0 clusters (***SYSTEM***)
1068 - Tried to rename an already deleted file
1069 - Printer not ready
1070 - Input time-out
1071 - End of Medium
1072 - Self test checksum failure
1073 - Must have at least one time-out block
1074 - Serial number of encrypted file does not match processor serial number
1075 - Can't find key in index
1076 - Key already exists in index
1077 - Key branching factor is too small
1078 - This copy of SDOS is not registered with Software Dynamics
1079 - Can't load file because last file loaded has different decryption key

1200 - SYSCALL Reply buffer not entirely within userspace
1201 - SYSCALL Write buffer not entirely within userspace
1202 - Reply buffer > 256 bytes for non read/write SYSCALL
1203 - Write buffer > 256 bytes for non read/write SYSCALL
1204 - Available channels exhausted--try again, later
1205 - Function not available under SDOS/MT
1206 - Not enough room to run SDOS/MT
1207 - Incorrect configuration for SDOS/MT
1208 - Interlock object already locked
1209 - No such object
1210 - Interlock object not locked
1211 - Interlock object was destroyed
1212 - Interlock object was forced to do RESET
1213 - Implementation limit reached
1214 - Illegal interlock function
1215 - SDOS/MT memory management failure (*** SYSTEM ***)
1230 - SDOS/MT already running
1231 - Terminal status for some port has changed
1232 - SDOS/MT primitives missing from configuration
1900 - Error detected after bringing drive on-line (Spin up)
1901 - Error detected after making drive off-line (Spin down)
1902 - Device name doesn't match media type/density
1903 - Device controller fault
1905 - VT driver is still busy with previous request
1906 - Another task is using this device
1907 - No room in reply buffer for activation character
1908 - CRT Field is wider than screen
1909 - Activation received
1910 - Timed Input period has expired
1911 - Specified device profile is not in this configuration
1912 - Device profile is not malleable

