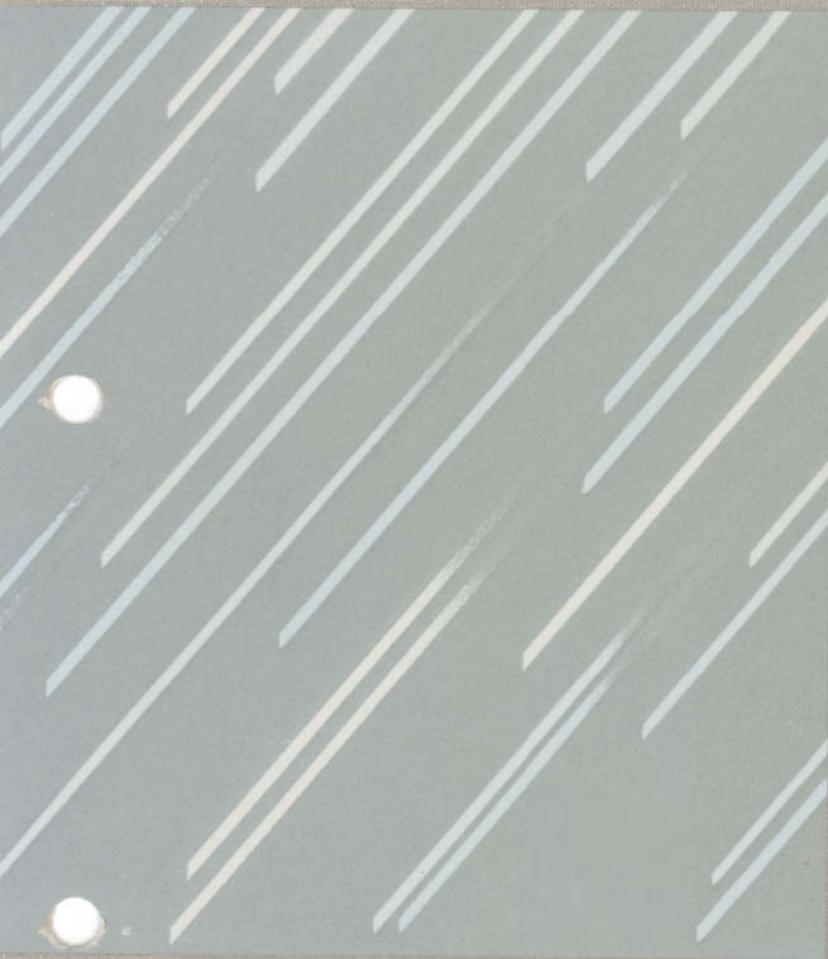


OS-9 Level Two



TERMS AND CONDITIONS OF SALE AND LICENSE OF TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND RADIO SHACK FRANCHISEES OR DEALERS AT THEIR AUTHORIZED LOCATIONS

USA LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

- A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment"), and any copies of software included with the Equipment or purchased separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.
- B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. LIMITED WARRANTIES AND CONDITIONS OF SALE

- A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. This warranty is only applicable to purchases of Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores, and Radio Shack franchisees and dealers at their authorized locations. The warranty is void if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or a participating Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.
- B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or Radio Shack dealer along with the sales document.
- C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.
- D. EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.
- E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

- A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE." IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE." NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNTO FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.
- B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.
- C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.
- D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the TANDY Software on one computer, subject to the following provisions:

- A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.
- B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.
- C. CUSTOMER may use Software on a multiuser or network system only if either, the Software is expressly labeled to be for use on a multiuser or network system, or one copy of this software is purchased for each node or terminal on which Software is to be used simultaneously.
- D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on one computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.
- E. CUSTOMER is permitted to make additional copies of the Software only for backup or archival purposes or if additional copies are required in the operation of one computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.
- F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.
- G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

- A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby Radio Shack sells or conveys such Equipment to a third party for lease to CUSTOMER.
- B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by Radio Shack.

VI. STATE LAW RIGHTS

The warranties granted herein give the original CUSTOMER specific legal rights, and the original CUSTOMER may have other rights which vary from state to state.

OS-9
Level Two
Operating
System

OS-9 Level Two Operating System (software):
© 1986, Microware Systems Corporation
Licensed to Tandy Corporation.
All Rights Reserved.

All portions of this software are copyrighted and are the proprietary and trade secret information of Tandy Corporation and/or its licensor. Use, reproduction or publication of any portion of this material without the prior written authorization by Tandy Corporation is strictly prohibited.

OS-9 Level Two Operating System (manual):
© 1986, Tandy Corporation.
All Rights Reserved.

Reproduction or use of any portion of this manual, without express written permission from Tandy Corporation and/or its licensor, is prohibited. While reasonable efforts have been made in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors in or omissions from this manual, or from the use of the information contained herein.

Tandy is a registered trademark of Tandy Corporation.

OS-9 is a trademark of Microware Systems Corporation.

BASIC09 is a trademark of Microware Systems Corporation and Motorola, Inc.

Getting Started

**Getting Started
With
OS-9**

About This Manual

Using Your OS-9 Handbook

If you feel that starting a new computer *operating system* is a “scary business,” relax. This handbook is designed to put you at ease when using OS-9. It is divided into two parts—each part has a different purpose.

What is in Part 1

“Part 1” of this handbook is designed to show you, step by step, how to set up and use your computer with OS-9. Follow the steps as they are described, and OS-9 is your obedient servant. The few instructions in “Part 1” are all that many OS-9 users ever need.

What is in Part 2

“Part 2” is for the more adventurous. OS-9 has an extensive repertoire of commands and functions to create and manage data and to make use of *peripherals* (devices you can connect to your computer, such as disk drives and printers). If you want to learn more about the operating system, and if you like to explore, “Part 2” is for you. You learn other useful OS-9 commands that prepare you to make use of all the functions and commands described in *OS-9 Commands*.



Contents

Part 1 / What You Need to Know About OS-9

Chapter 1 What is an Operating System?	1-1
Instructing Your Operating System	1-1
Using Application Programs and Computer Languages	1-2
Using Peripherals	1-3
Why Use OS-9?	1-4
How Much Do You Need to Know About OS-9? ..	1-5
Chapter 2 How to Start and Exit Your System ..	2-1
Booting OS-9	2-2
Rebooting OS-9	2-3
Exiting OS-9	2-3
Upper- and Lowercase Characters	2-4
OS-9 Error Messages	2-4
Chapter 3 What You Need to Know to Use Floppy Drives	3-1
Write Protection for Diskettes	3-2
Disk Drive Names	3-2
Making Copies of Diskettes	3-3
Formatting With One Disk Drive	3-3
Formatting With Two Disk Drives	3-4
Using the Backup Command	3-5
Making Copies With One Disk Drive	3-5
Making Copies With Two Disk Drives	3-7

Part 2 / Organization, Commands, and Keys

Chapter 4 Files and Directories	4-1
About Files	4-1
About Directories	4-1
Multiple Directories	4-4
About File and Directory Names	4-4
Examples of Filenames	4-4
About Pathlists	4-5
Anonymous Directory Names	4-6
About Device Names	4-6

Chapter 5	Commands and Keys	5-1
	Typing Commands	5-1
	Editing Commands	5-1
	Command Parameters	5-2
	Using Options	5-2
	Using Commands	5-3
	Accessing Commands	5-4
	Commands from Disk	5-5
	Changing the Execution Directory	5-6
	Changing the Data Directory	5-7
	Changing System Diskettes	5-7
	Video Display and Keyboard Functions	5-8
	Special Keys	5-8
Chapter 6	OS-9 Toolkit	6-1
	Viewing Directories	6-1
	Creating Directories	6-1
	Deleting Directories	6-2
	Displaying Current Directories	6-2
	Copying Files	6-3
	Deleting Files	6-4
	Renaming Files	6-4
	Looking Inside Files	6-5
	Loading Command Modules into Memory	6-5
	Listing the Command Modules in Memory	6-5
	Deleting Modules from Memory	6-6
	Using Other Commands	6-6
Chapter 7	Customizing Your System	7-1
	Creating a New System Diskette	7-1
	Monitor Types	7-8
	Using Windows	7-9
	Establishing a Window	7-9
	Changing Window Colors	7-11
	Eliminating a Window	7-12
	Using Startup to Establish a Window	7-13

Index

Part 1

**What You Need to Know
About OS-9**



What is an Operating System?

OS-9 is a disk *Operating System* (that's what OS stands for). An operating system is a group of programs acting as a message center and an interpreter. Using your instructions, an operating system manages the computer's working circuits.

In fact, thinking of OS-9 as your computer manager is helpful. The boss (that's you) gives orders. OS-9 (the manager) sees they get done.

To operate OS-9 you need at least one floppy disk drive attached to your computer. OS-9 is originally configured to recognize two floppy disk drives. Later, this handbook describes how to let OS-9 know if you have more than two floppy disk drives, or if you have other hardware (printers, modems, hard disks, and so on) you want it to recognize.

Instructing Your Operating System

You give your commands to OS-9 by typing them. Because OS-9 does exactly (and only) what you tell it, your entries must be precise and have perfect *syntax* (spelling and form). You must also be sure to give OS-9 every detail it needs to perform a task.

For instance, if you told your office manager to, "Make a phone call," what can the manager do? Obviously, not much that is helpful to you. The manager must know who to call, the phone number, and what to say. OS-9 is the same. It must have all the details before it can carry out your commands properly.

To show you how to instruct your operating system, the handbook asks you to type characters, words, and lines on your keyboard. When you do, you are issuing *commands* to OS-9. Technically, a command is only one word that describes the action you want OS-9 to perform. A *command line* is a command with all of its qualifiers.

In this manual, command lines usually contain words in boxes, such as ENTER. These indicate keys that you press.

The manual also asks you to **press** key sequences. For instance, when asked to press CTRL C, hold down the key marked CTRL, and while holding down CTRL, press C.

Characters that are not in boxes are typed individually. For instance, if you are asked to type the command line `format /d0` (ENTER), press each key individually (F O R M A T / d 0 ENTER).

If you make a mistake while typing, use  to move back to the error. Then retype that portion of the line.

Using Application Programs and Computer Languages

A computer *application* is a program designed to accomplish specific tasks. There are application programs to help you write letters or documents (word processors), keep a mailing list (data managers), and keep financial records (accounting packages). There are also programs to help you study for a test, play a game, play music, draw a picture, and much more.

Such application programs usually require that you use OS-9 to start your computer. A few application programs let you start directly from the application diskette. Different programs can require different procedures, and you should check your application program's documentation for specific instructions.

Application programs have special screen displays and *menus* to instruct you, or that require you to perform a particular action, such as press a key. When you are operating from an application program, that program passes your instructions to OS-9. OS-9 manages the computer's operations in the background, and its functions are invisible to you.

You can also use computer *languages* to write your own application programs. BASIC is a language. If you read the *Color Computer Disk System* manual, you already know a bit about it. There are languages you can purchase to use with OS-9 to create programs, such as assembly language, Pascal, C, and BASIC-09.

Like application programs, each language has its own startup method. The manuals that come with the languages tell you how to get them running on your Color Computer 3.

Using Peripherals

OS-9 lets you control much more than your computer's operations. It also gives you control over other hardware devices such as disk drives, a printer, modems, windows, other terminals, and so on.

Each device has a "System Name," an abbreviation preceded by a slash (/). OS-9 can only recognize a device if you type its name exactly as shown below. See Chapter 7, "Customizing Your System" for information on how to tell OS-9 what devices you want it to handle.

System Name	Description
/P	A printer connected through your computer's RS-232 port. The RS-232 port is a <i>serial</i> port, and you must have a printer with a serial connection, such as the Radio Shack® DMP 430.
/T1	A data terminal or another computer acting as a terminal, connected through the RS-232 port of your computer. If you are using another computer as a terminal, it must run a terminal program that makes it perform as a terminal.
/T2	Another data terminal or another computer acting as a terminal, connected to an optional RS-232 communications pak in a Multi-Pak Interface. If you are using another computer as a terminal, it must run a terminal program that makes it perform as a terminal.
/T3	Another data terminal or another computer acting as a terminal, connected to the optional RS-232 communications pak in a Multi-Pak Interface. If you are using another computer as a terminal, it must run a terminal program that makes it perform as a terminal.
/M1	A modem using an optional 300-baud modem pak in the optional Multi-Pak Interface. A modem allows you to communicate with other computers either directly or over phone lines.

System Name	Description
/M2	Another modem using an optional 300-baud modem pak in the optional Multi-Pak Interface.
/D0	A floppy disk drive.
/D1	Another floppy disk drive
/W, /W1, /W2, /W3 /W4, /W5 /W6, /W7	Windows that you can establish on your OS-9 system. You use CLEAR to <i>page</i> among windows you create. See "Using Windows" in Chapter 7 and <i>OS-9 Windowing System Owner's Manual</i> for information on creating windows.

Why Use OS-9?

You now know that OS-9 is an operating system for your Color Computer. You might also have heard that, in the world of computer operating systems, OS-9 is a leader. Perhaps that is why you bought it. OS-9 stands out for several reasons. Some of its strong points are:

- *File* managing capabilities.
- *Multi-user* features. With OS-9, more than one person can use the same computer at the same time.
- *Multi-tasking*. OS-9 can handle several jobs at the same time.
- *Window* functions that let you divide your display screens into sections in which you can have one or more operations running, all at the same time.
- *Input/Output* capabilities. OS-9 can communicate with TVs and monitors, disk drives, printers, and other computers.
- A sophisticated repertoire of commands.
- Sophisticated programming languages.

If you are not familiar with such terms as files, multi-user, multi-tasking, and commands, don't worry. The handbook explains these terms and more.

Programmers like OS-9 because of its powerful features. It lets them show off all of their skills. As a result, another OS-9 feature is the wide range of excellent programs that you can use with the system.

How Much Do You Need to Know About OS-9?

You might wonder how much you really need to know to use OS-9. The answer varies with your needs, and with the application programs you intend to use.

However, regardless of how you intend to use your computer, there are some OS-9 procedures you must know. For instance, you must know how to load OS-9, how to prepare diskettes to store data, and how to make copies of data or entire diskettes. This part of your handbook makes these jobs easy.

Regardless of how careful you are, there are times when things go wrong. When this happens, OS-9 displays an *error message* on the screen. This part of the manual also helps you to understand error messages and what to do about them.

How to Start and Exit Your System

Starting your computer and initializing an operating system is called *booting*. In a sense, the computer is pulling itself up by its bootstraps.

To run OS-9, Level II, you must have a Color Computer 3 with at least one floppy disk drive. Your OS-9 system diskette includes modules to support the following Color Computer hardware:

- Up to 512K RAM
- A Keyboard
- An Alphanumeric Video Display
- A Color Graphics Display
- Floppy Disk Drives (one or two)
- Joysticks (one or two)
- A Serial Printer
- An RS-232C Communications Port

If you connect a Multi-Pak Interface to your computer, and use the CONFIG utility from your BASIC09/CONFIG diskette (see Chapter 7), OS-9 can support the following devices:

- As many as two external RS-232 communications cards
- As many as two modem paks
- As many as two additional floppy disk drives

Note: The Multi-Pak Interface has four cartridge slots. A floppy disk controller must be in Slot 4. You can put modem paks, or RS-232 paks in Slots 1, 2, or 3.

Booting OS-9

Use the instructions in the *Color Computer Disk System* manual to turn on your computer system. After you do, the video screen displays a copyright message followed by the letters, `OK`. This is Disk Extended Color BASIC's way of telling you that it is ready to get to work. It is waiting for your commands.

To load OS-9, follow these steps:

1. Insert the OS-9 System Master diskette into Drive 0.
2. At the `OK` prompt, type:

```
DOS 
```

OS-9 starts. If the DOS command returns a syntax error (`SN? ERROR`), be sure you entered the command correctly. If DOS still returns the error, check to make sure you have installed your disk cartridge properly.

3. After OS-9 displays its startup message, this prompt appears:

```
yy/mm/dd hh:mm:ss  
Time?
```

4. Type the year, month, date, hours, minutes, and seconds in the format requested; then press . For instance, if the date and time is September 3, 1986, 1:22 p.m., type:

```
86/09/03 13:22 
```

Note that the time is entered in 24-hour notation and that the seconds (:SS) are optional.

You can bypass this time and date prompt by only pressing . However, if you do, OS-9 cannot provide the correct date when you create and save data on disk. Also, it cannot provide the correct date and time for application programs that require them.

After you enter the date and time, the OS-9 prompt appears and OS-9 is now in control and ready to accept a command.

You should always keep the OS-9 System diskette in Drive 0 (/D0) while running OS-9 unless you have a hard disk containing your system files. An OS-9 System diskette is a backup copy of the OS-9 System Master diskette. The instructions for making copies are in the next chapter.

Rebooting OS-9

If you need to reboot OS-9 after the initial startup, press your computer's reset button (located at the right rear of the computer). Pressing the reset button one time causes the OS-9 boot message to reappear. The system then loads as it did originally. Be sure the System Master diskette is in Drive /D0 when you reboot.

Pressing the reset button twice returns the computer to Disk BASIC.

Exiting OS-9

In the same manner that you use OS-9 to start operations, you should use OS-9 to exit or close operations. For instance, if you are in the middle of a process, it is unwise to suddenly turn off your computer. Doing so can destroy files or garble disks.

You can usually terminate an operation by pressing **BREAK** or **CTRL** **E**. In some instances, you must let an operation complete its function before you can regain control of OS-9. If you are using an application program, that program's manual tells you how to exit the program to the OS-9 command level.

You should always be at the OS-9 command level to turn off your computer. Then follow these steps:

1. Be sure the OS-9 system prompt and cursor are displayed.

Note: You can *turn off* the OS-9 cursor. If you or an application program has done so, the cursor does not display at the command level.

2. Take out any floppy diskettes from the disk drives, put them back in their protective envelopes, and store them in a safe place.
3. Turn off all the equipment attached to your computer such as a printer or disk drive(s); then turn off your TV or monitor. Last of all, turn off your computer and Multi-Pak Interface (if you have one). If you plug your equipment into a power strip, you can use the power strip switch to turn off all equipment at one time.

Upper- and Lowercase Characters

OS-9 can display both upper- and lowercase letters. However, you can tell it you want to use only uppercase. To do this, type:

```
tmode upc [ENTER]
```

If you do this, you cannot type lowercase letters, and the system displays all uppercase letters. To switch back to both uppercase and lowercase, type:

```
tmode -upc [ENTER]
```

Even when you are in the upper-/lowercase mode, you can switch to typing all uppercase by pressing **CTRL** **O**. Everything you type is now uppercase, but the computer can display both upper- and lowercase. Press **CTRL** **O** to switch back to upper-/lowercase.

If you want to type only one uppercase letter, hold down **SHIFT** while you press that letter.

It does not matter to OS-9 whether you type in uppercase or lowercase letters, or any combination of upper- and lowercase letters. For instance, instead of typing `TMODE UPC`, you can type `tmode upc` or `Tmode UPC`.

OS-9 Error Messages

Everyone makes a mistake now and then when typing commands. If you type something the operating system doesn't recognize, or if you ask it to do something it cannot do, it displays an *error message*. This message is a number that refers to the type of problem that OS-9 has encountered. For instance, if you type `xxxx` **[ENTER]** (which is nonsense to OS-9), the system displays:

```
ERROR #216
```

If you don't know the meaning of the system error number you have two options: (1) you can look up the reference in *OS-9 Commands* under Appendix A, "Error Codes" or, (2) you can type:

```
ERROR 216 [ENTER]
```

Either method shows you that Error #216 means "Path Name Not Found." OS-9 thought you wanted it to execute a command but it could not find one named `xxxx`.

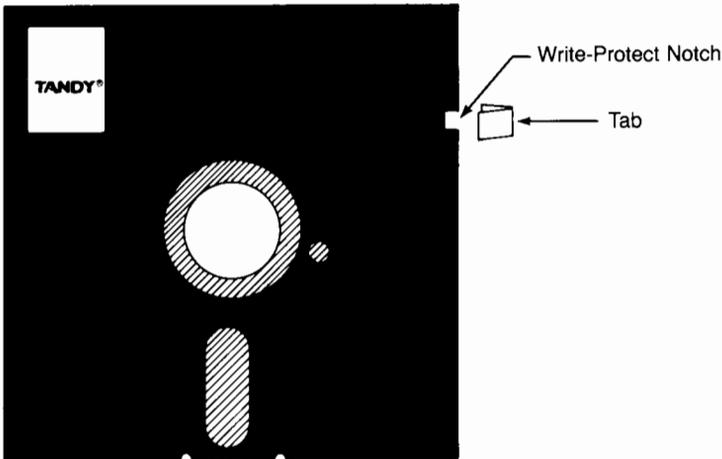
Other OS-9 error messages tell you if you have used all of a disk's storage space, if the computer's memory is full, if you try to create two files with the same name, and so on.

What You Need to Know To Use Floppy Drives

Floppy diskettes require careful handling. You might already be familiar with how to take care of diskettes from reading your *Color Computer Disk System* manual. If not, or as a reminder, review the following points:

- Always make copies of important diskettes. The price of a diskette is small compared to the time it can take to replace destroyed data.
- Copy data you are working with regularly. If you experience a power failure while using your computer, the data on any diskettes you have in a drive can be destroyed. Other accidents can happen as well.
- Always keep the protective paper or cardboard envelope on your diskette when it is not in use.
- Your drive accesses a diskette through the oblong slot in the diskette's jacket. **Never** touch the diskette through this hole. The oil from even the cleanest hand can destroy data, making the diskette useless.
- Do not bend diskettes.
- Store diskettes away from excessive heat, dust, and any magnetic source. Even components in disk drives, video displays, TVs, and electric motors can garble the data on diskettes.
- If you must write on a diskette label after placing it on the diskette, use only a soft felt pen.
- Do not switch your computer, disk drive(s), or Multi-Pak interface on or off while you have a diskette in a disk drive.

Write Protection for Diskettes



Most diskettes have a square notch cut from one corner. This is a *write protect* notch. If you place a special adhesive tab (supplied with diskettes) over both sides of this notch, your computer can no longer write (store) data on it. This feature protects diskettes from inadvertent destruction of data.

Removing the tab again lets you write data onto the diskette.

Disk Drive Names

OS-9 has its own method of referring to your disk drives. What your *Color Computer Disk System* manual calls Drive 0, OS-9 calls Drive /D0. This is your first drive if you have more than one floppy disk drive connected to your system. Subsequent drives are named /D1, /D2, and so on.

If you have a hard disk attached to your system, OS-9 refers to it as Drive /H0. A second hard disk drive is named /H1.

Making Copies of Diskettes

Before you can store information on a diskette, you must *format* it. Formatting is the process of magnetically arranging a disk's surface so that OS-9 can store and locate information. The following steps tell you how to format a diskette. Format at least two diskettes at this time to use in making backups (copies) of your two OS-9 system diskettes. If you have other important diskettes to backup, format as many diskettes as you require.

Formatting With One Disk Drive

1. If you have not already done so, place a write-protect tab on your System Master diskette. Then, turn on and boot your computer as described in Chapter 2.
2. With the OS-9 System Master diskette in your drive, type:

```
load format .
```

3. Select a diskette that does not contain data or that contains data you do not want to keep. Make sure it does not have a foil tab covering the write-protect notch. Put it in your disk drive (Drive /D0) in place of your OS-9 System Master diskette and type:

```
format /d0 
```

The following prompt appears:

```
COLOR COMPUTER FORMATTER  
Formatting drive /d0  
y (yes) or n (no)  
Ready?
```

4. Press to begin formatting. OS-9 asks you for a Disk Name:. Type any name, using a maximum of 32 characters. For example, you can type to name the diskette "s."

Next OS-9 verifies that the diskette is formatted properly. The screen shows each *track* number in hexadecimal notation during verification. A track is a concentric ring around the diskette on which information is stored.

5. When formatting is complete, OS-9 shows you the Number of good sectors. This number depends on the type of disk drive you are using. For a 35 track, single-sided drive, the number should be \$000276 (hexadecimal 276 sectors). The OS-9 prompt and cursor reappear. Remove the newly formatted diskette from the drive, and store it in a safe place until you are ready to use it.

Format as many diskettes as you need by following Steps 3 through 5.

Formatting With Two Disk Drives

1. If your computer is off, turn it on, and boot OS-9 as outlined in Chapter 2.
2. At the system prompt (OS9:), type `format /d1` . The screen shows:

```
COLOR COMPUTER FORMATTER
Formatting drive /d1
y (yes) or n (no)
Ready?
```

3. Insert a blank disk, or one which does not contain data you want to keep, into Drive /D1, and close the latch. Be sure the diskette does not have a foil tab covering the write-protect notch. Press .
4. OS-9 formats the diskette; then asks you for a Disk Name:. Type any name, using a maximum of 32 characters. For example, you can type `s` to name the diskette "s."

Next OS-9 verifies that the diskette is formatted properly. The screen shows each *track* number in hexadecimal notation during verification. A track is a concentric ring around the diskette on which information is stored.

5. When formatting finishes, OS-9 shows you the Number of good sectors. This number depends on the type of disk drive you are using. For a 35-track, single-sided drive, the number should be \$000276 (hexadecimal 276 sectors). The OS-9 prompt and cursor reappear. Remove the newly formatted diskette from the drive, and store it in a safe place until you are ready to use it.

Format as many diskettes as you need by following the same procedure.

Using the Backup Command

BACKUP is one OS-9 command that you can expect to use frequently. It is the command you use to make copies of your diskettes. **We strongly recommend that you now use the following instructions to make copies of your OS-9 system diskettes.** You can only copy diskettes that are created in the same type of disk drive you are using. Your OS-9 system diskettes are 35 track, single sided.

BACKUP uses two terms you need to understand. They are *source* and *destination*. A source diskette is the diskette that contains the program, file or data that you want to backup. The destination diskette is the blank formatted diskette you prepared to receive the copied data.

Note: Some application programs you buy do not let you make copies of their diskettes. Check the program manual for information on protecting the data on these diskettes.

Making Copies With One Disk Drive

1. If your computer is off, turn it on, and boot OS-9 as outlined at the beginning of Chapter 2.
2. At the system prompt (OS9:), type:

```
backup /d0 #32K 
```

This tells OS-9 to make a backup of the diskette in Drive /D0. The screen displays the following prompt:

```
Ready to backup from /d0 to /d0  
? :
```

3. Leave the System Master diskette in Drive /D0 to make a backup of it. To back up one of your other diskettes, for example the BASIC09/CONFIG diskette, remove the System Master diskette and replace it with the diskette you want to copy.
4. Press when you are ready to continue. The screen displays:

```
Ready Destination, hit a key:
```

5. Replace the source diskette with the destination diskette. Then, press the space bar to continue BACKUP.

When you back up one diskette to another, any data previously existing on the destination diskette is *overwritten* (destroyed). OS-9 gives you a chance to make sure you have inserted the proper destination diskette by displaying the message:

```
DISK NAME
  is being scratched
Ok ?:
```

“Scratched” means that OS-9 is going to replace any data on the diskette with new data from the source diskette. BACKUP also gives the destination diskette the same name as the source diskette—the destination becomes a duplicate of the source.

6. Press to keep going. The screen asks you to:

```
Ready Source, hit a key:
```

7. Remove the formatted diskette from Drive /D0, and replace it with the source diskette that contains the data you want to copy. Press the space bar.

In a moment, a prompt asks you to:

```
Ready Destination, hit a key:
```

8. Remove the source diskette and replace it with the destination diskette. Press the space bar.
9. Continue switching diskettes as the screen instructs you until you see:

```
Sectors copied: $0276
Verify pass
```

Followed in a moment by:

```
Sectors verified: $0276
OS9:
```

The diskette now in your drive, the destination diskette, is a duplicate of the source diskette. If you copied the System Master or the BASIC09/CONFIG diskette, store it in a safe place, and use the copy as your *working* diskette. Reserve the original diskette for making future backups.

Note: For computers with 512K of memory, OS-9 can backup a diskette faster if you replace #32K in Step 2 with #56K.

Making Copies With Two Disk Drives

1. If your computer is off, turn it on, and boot OS-9 as outlined at the beginning of Chapter 2.
2. At the system prompt (OS9:), type:

```
backup /d0 /d1 #32K 
```

This tells OS-9 to make a backup of the diskette in Drive /D0.

The screen displays the following prompt:

```
Ready to backup from /d0 to /d1  
?:
```

3. Leave the System Master diskette in Drive /D0 to make a backup of it. To back up one of your other diskettes, for example the BASIC09/CONFIG diskette, remove the System Master diskette and replace it with the diskette you want to copy.
4. Press when you are ready to continue.

When you back up one diskette to another, the process *overwrites* or destroys any data previously existing on the destination diskette. OS-9 gives you a chance to make sure you have inserted the proper destination diskette by displaying the message:

```
DISK NAME  
is being scratched  
ok ?:
```

“Scatched” means that OS-9 replaces any data on the destination diskette with new data from the source diskette. As well, BACKUP gives the destination diskette the same name as the source diskette—the destination becomes an exact duplicate of the source.

6. Press Y to keep going.

Copying continues. When the procedure ends, you see:

```
Sectors copied: $0276  
Verify pass
```

Followed in a moment by:

```
Sectors verified: $0276  
OS9:
```

The diskette in Drive /D1 is now a duplicate of the source diskette. If you copied the System Master or the BASIC09/CONFIG diskette, store it in a safe place, and use the copy as your *working* diskette. Reserve the original diskette for making future backups.

Note: For computers with 512K of memory, OS-9 can backup a diskette faster if you replace #32K in Step 2 with #56K.

Part 2

**Organization, Commands,
and Keys**

Files and Directories

Before you can use OS-9 extensively, you need to know how the system organizes and stores data on disk. The information in this section is true for both floppy diskettes and hard disks. However, because of the greater storage capacity of a hard disk, it is of particular importance to hard disk users.

About Files

Consider the information stored on disks to be of two basic types, programs and data. A program is *code* that causes your computer to execute a task. Data is information that a program uses or that a program creates.

All the information that OS-9 stores on disks, whether program or data, is stored in units called *files*. Whenever a program creates a file, OS-9 defines a portion of your disk to store it. It keeps the location of the file in a special list (called a *directory*), also located on the disk, so that it knows where to find your program or data the next time you want it.

About Directories

A directory is a storage space for filenames, other directory names, or both.

After you format a disk, it contains one directory called the ROOT directory. However, a disk can have many directories. For instance, besides the ROOT directory, your System Master diskette contains the CMDS and SYS directories. The ROOT and CMDS directories are especially important to you.

When you boot OS-9, you automatically begin operation from these two directories. The ROOT directory becomes your current *data directory* and the CMDS directory becomes your current *execution directory*.

Whenever you ask OS-9 to store a file on a diskette, it automatically stores it in the current data directory (the ROOT directory), unless you tell it otherwise. If you ask OS-9 to execute a command or program, it automatically looks for that command or program in the execution directory (the CMDS directory), unless you tell it otherwise.

Every OS-9 directory can also contain other directories, called *subdirectories*. For instance, SYS, and CMDS are established as subdirectories of the ROOT directory. Put in chart form, your ROOT directory with its subdirectories looks like this:



Figure 4.1

But there are also files in the ROOT directory, OS9Boot and Startup are two. The full ROOT directory might look like this:

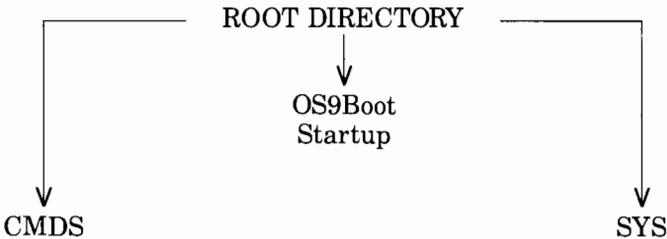


Figure 4.2

You can create another subdirectory of the ROOT directory if you want. For instance, if you created a directory named FAMILY, the chart of the ROOT directory looks like this:

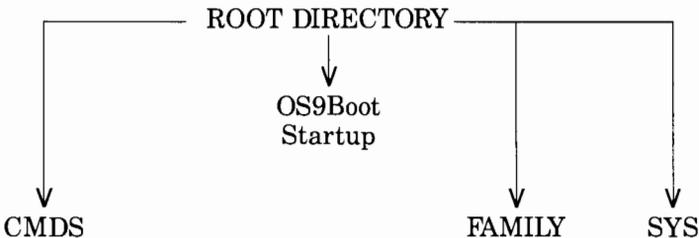


Figure 4.3

After you create the FAMILY directory, you can also create other directories in it. Suppose you create two subdirectories named PLEASURE and WORK. The chart organization is as follows:

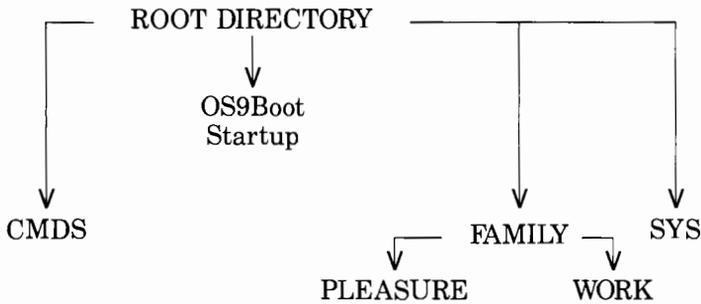


Figure 4.4

The directories you create also can hold files. If you created three files each in the PLEASURE and WORK directories, the chart might look like this:

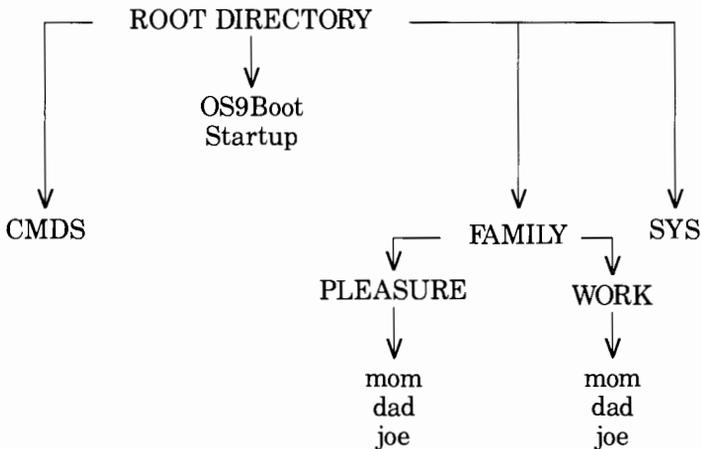


Figure 4.5

You can continue to create files and subdirectories in any or all of your disk's directories until you fill the disk's storage space.

Multiple Directories

There is nothing wrong with storing all your files in the ROOT directory. Doing so makes it easy to access them because they are always in your data directory.

However, creating multiple directories makes it easy to keep your data organized when you have many files, or if more than one person is using the same disk. Such a multiple-directory organization is especially helpful when using hard disks, which can store hundreds of individual files.

Also, when you have multiple directories, you can store files having the same name in different directories without conflict, such as in the PLEASURE and the WORK directories of Figure 4.5.

About File and Directory Names

The file and directory names shown so far consist only of letters of the alphabet, but you can use other characters and symbols in a file or directory name as long as each name begins with a letter. The following is a complete list of acceptable characters:

- Uppercase letters: (A-Z)
- Lowercase letters: (a-z)
- Decimal digits: (0-9)
- The underscore character (___) and the period (.)

You can include as many as 29 characters in a file or directory name.

Examples of Filenames

The following are samples of filenames that OS-9 can recognize:

mydata	samfile
mydata1	Dollar_gifts
records.srt	help.file
XXX.xx	file#1.txt
prog1.bas	program.sourcecode
prog2.bas	program.opcode

Examples of invalid filenames are:

- his*hersbecause * is not a valid character for names
- .DATAbecause the name does not begin with a letter
- COST+INTbecause + is not a valid character for names
- 100_dollar_gifts ...because names cannot begin with a digit

About Pathlists

Because you can organize OS-9 disks into multiple levels, you need a way to tell the system where to find directories and files. The directions you give are called *pathlists*.

A pathlist is exactly what its name implies—a path (or route) to the device, directory, or file you want to access. For instance, if you are in the ROOT directory (see Figure 4.5) and want to look at the contents of a file in the WORK directory, you must tell OS-9 how to get there. The pathlist from the ROOT directory to the Dad file is `family/work/dad`. OS-9 expects you to separate the junctions of pathlists with slashes. To look at the contents of Dad, you type:

```
list family/work/dad 
```

Because you are accessing a file on the current disk, you do not need to specify a drive name. Because every disk contains a ROOT directory, and all other directories and files branch from it, ROOT is always implied in a pathlist. If Figure 4.5 represented the diskette in Drive /D1, the pathname to the Dad file would be `/d1/family/work/dad`.

Depending on the location of the directory or file you want to access, a full pathlist need not contain any more than the name of a drive, the name of a directory, or the name of a file. For instance, the complete pathlist from the ROOT directory of Figure 4.5 to the Startup file is `startup`. To look at the contents of Startup, type:

```
list startup 
```

Anonymous Directory Names

To save time, or if you do not know a full pathlist, you can refer to the current directory, or to a higher-level directory, using an *anonymous* name, or name substitute, as follows:

- One period (.) refers to the current directory
- Two periods (..) refer to the *parent* of the current directory (the next highest-level directory).
- Three periods (...) refer to the directory two levels up, and so on.

You can use an anonymous directory name in place of a pathlist or as the first name in a pathlist. Some examples are:

`dir ..` lists names in the current data directory's parent directory.

`del ../temp` deletes the file called Temp from the current data directory's parent directory.

Anonymous names can refer to either execution or data directories, depending on the context in which you use them.

About Device Names

In the same manner that OS-9 has names for its commands, it also has names for its devices. These names are abbreviations of actual device names. For instance, instead of typing Disk Drive 0 to refer to your first disk drive, you only need to type /D0. To refer to your printer, type /P. OS-9 windows are named /W through /W7.

All of OS-9's device names are preceded by a slash—this is how OS-9 can tell you are referring to a device rather than a directory or file. When you purchase your System Master diskette, OS-9 is configured to recognize two disk drives, a printer, and one terminal port. For information on how to configure your system to recognize other devices, see Chapter 7.

Commands and Keys

You already put OS-9 to work with commands such as FORMAT and BACKUP. In these cases the manual told you exactly what to do to accomplish a very specific task. If you want to strike out on your own, you should know some additional background information.

Typing Commands

As explained earlier, some OS-9 files are programs. You tell OS-9 to execute these programs by typing the program (file) name and pressing **ENTER**. You are then issuing a command to OS-9. That's all a command is, the name of a program for the system to execute. The following are some rules about commands:

- You can enter a command whenever the screen displays the system prompt (OS9:).
- A command consists of one word, the command name. A *command line* consists of one or more command names and their associated *parameters* and *modifiers*. Parameters and modifiers are special information you include with a command that provide necessary data for the command to operate, or that affect the command's operation.
- A command line can have a maximum of 198 characters including any combination of upper- or lowercase letters. To execute a command, press **ENTER**. For example, to clear the screen, type:

```
display 0c ENTER
```

Editing Commands

OS-9 is very particular about the commands you type. If you make any mistake, OS-9 either does not understand (and tells you so with an error message) or does the wrong thing.

If you see that you made a mistake before you press **ENTER**, you have two choices: (1) use **←** or **CTRL H** to move the cursor to the mistake, and retype that portion of the line, or (2) press **CTRL X** or **SHIFT ←** to erase the line you are typing, and start over.

Command Parameters

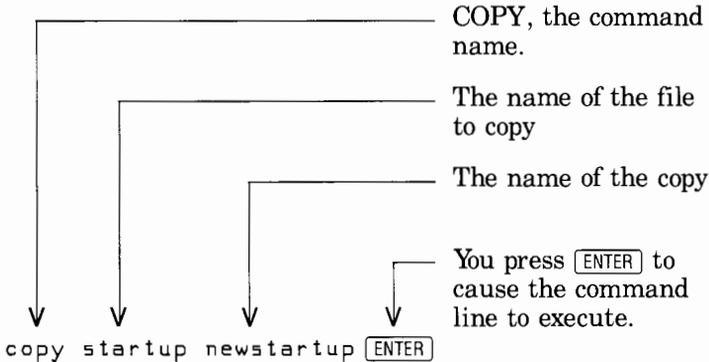
You can follow a command name with one or more parameters that give OS-9 more specific instructions. For example, in the command line:

```
list file1 
```

LIST is the name of the command that displays the contents of a text file. *File1*, the specified parameter, is the name of the file that you want displayed.

Note: In a command line, always use spaces to separate parameters from their command, and from each other. Parameters cannot contain spaces. Chapter 6 discusses parameters for each OS-9 command.

Some commands have more than one parameter. For instance, COPY requires two parameters: the name of the file being copied, and the name of the new file you want COPY to create. If you want to copy a file called Startup, and call the copy Newstartup, your command line reads:



Using Options

Command lines can also contain another type of parameter, called an *option*. An option changes the way a command performs. For instance, the command DIR, without parameters, shows the name of all files in the current data directory.

However, if you add the E option as a parameter to the command, like this:

```
dir e 
```

the output includes not only the names of the files, but also complete statistics about each file—the date and time created, size, security codes, and so forth.

To display complete information about each file in SYS, type:

```
dir sys e 
```

Using Commands

As described in Part 1, OS-9 acts in much the same manner as an office manager. It looks after the operation of your computer and equipment. Because OS-9 is only a manager, it expects you to make the necessary decisions.

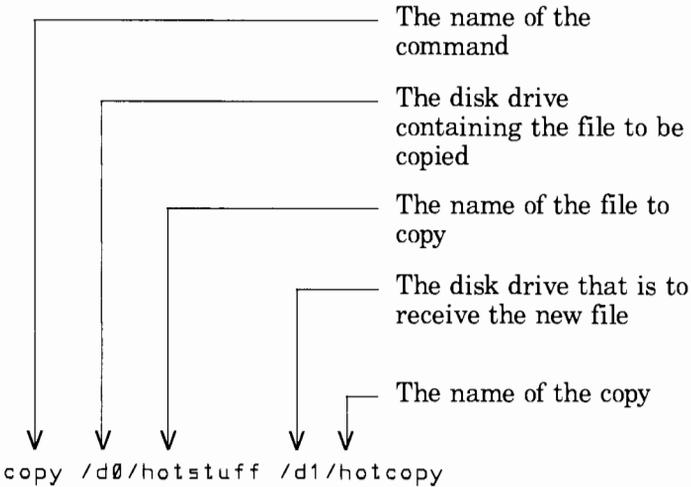
For example, suppose you have an important file named Hotstuff that you want to copy. Before giving it to your office manager (OS-9), you must make executive decisions, such as:

- Do you want the copy on disk, paper, or the computer screen?
- If you want the copy on disk, which disk?
- If you want the copy on the same disk, what name do you want to give the second copy so OS-9 is not confused?
- If you want the copy on the computer screen, do you want the display to pause when it fills the screen?

You make the decisions, OS-9 manages the job. For instance, if your decision is to copy Hotstuff from one diskette to another, you might type the following command line:

```
copy /d0/hotstuff /d1/hotcopy 
```

This is how OS-9 sees your command:



This command line tells OS-9 to copy a file named Hotstuff from your floppy disk Drive /D0 to a second floppy Drive /D1. The file copy is given the new name, Hotcopy.

You only need to know the name of the file you want to copy, on which disk it is located, and the disk on which you want the new copy. OS-9 manages the operation for you.

Accessing Commands

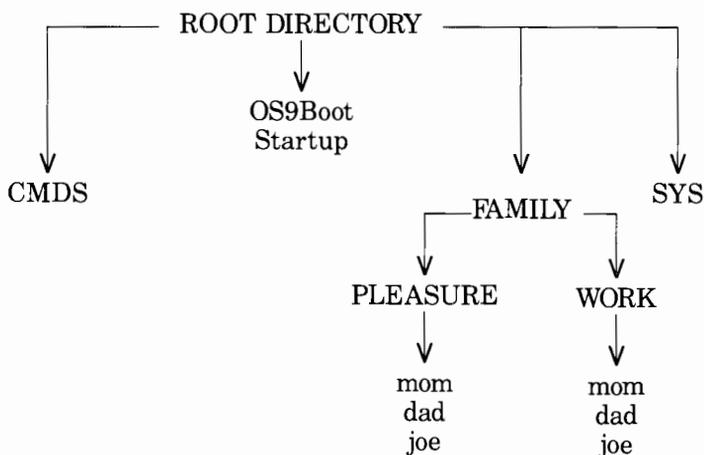
OS-9 has two ways to access commands. Some commands reside on a disk. When you type the command name and press `ENTER`, OS-9 must look on the disk, load the program into the computer's memory, and then execute it.

Other commands are loaded into your computer's memory at startup, or you can load them into memory later. When you call a command that is in memory, it is executed immediately. There is no delay while OS-9 finds it on disk.

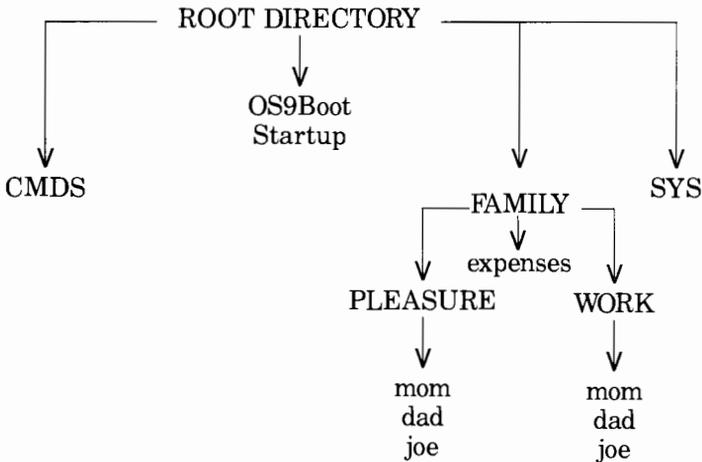
Commands from Disk

When you give OS-9 a command that it cannot find in memory, it looks for the command in the current execution directory. If it cannot find it there, it checks the current data directory. If it still cannot find it, the system issues Error Message #216, Path Name Not Found. If the command you want executed is in a directory other than the current directory, you must tell OS-9 where to find it. Remember, when initialized, OS-9 sets the CMDS directory of the system disk to be the execution directory.

For instance, suppose you booted your system using a diskette configured like the example we used in Chapter 4:



When the system starts, the ROOT directory is the data directory, and the CMDS directory is the execution directory. Now, suppose you had a program named Expenses in the family directory:



(Remember that a program and a command are really the same thing.)

You can now access (use) the expenses program in two ways. One way is to specify a pathlist from the ROOT directory to execute Expenses, such as:

```
/d0/family/expenses 
```

Another way is to change the execution directory.

Changing the Execution Directory

To change the execution directory to the FAMILY directory, type:

```
chx /d0/family 
```

Or specify a pathlist relative to the current execution directory, such as:

```
chx ../family 
```

To execute the Expenses program, you now only need to type `expenses` .

However, after you change the execution directory, to use a command in the COMMANDS directory, you must tell OS-9 where to find it. For example, to format a new diskette in Drive /D1, type:

```
/d0/cmds/format /d1
```

Changing the Data Directory

Suppose that the Expenses program keeps track of work and pleasure expenses for Mom, Dad, and Joe. Unless you tell OS-9 otherwise, it looks for data files in the current data directory, the ROOT directory. To tell OS-9 to look for data files in the PLEASURE directory, type:

```
chd family/pleasure 
```

The slash between FAMILY and PLEASURE tells the system that PLEASURE is a branch of FAMILY. Subordinate directories and files are always separated from their parent in this way.

Now, when Expenses needs data, it knows to look in the PLEASURE directory.

Changing System Diskettes

Although it is preferable to leave the system diskette in place while the system is running, particularly with multiuser systems, there might be times when you need to use another diskette. Only remove the current diskette when the screen displays the OS-9 prompt, followed by the cursor. If you do remove the system diskette and begin to use another one, use the CHD and CHX commands to tell OS-9 where you want to be located on the new diskette. (For directions, see Chapters 2 and 6.) Those commands set both directory pointers, data and execution, for the new diskette.

While using a program or command, do not remove a diskette and insert another unless the program or command asks you to. You can lose data, or entire files, if you do.

Video Display and Keyboard Functions

OS-9 has many features that expand the capability of the Color Computer's video display and keyboard.

- The video display has upper-/lowercase, screen pause, graphics functions, and 80 column displays if you have a monitor connected.
- The **ALT** key provides an alternate key function. Holding down **ALT** while pressing another key sets the *high order bit* of the character pressed. That is, it adds 128 to the normal ASCII value produced by that key. Holding down **ALT** while pressing any other key produces a graphics character on the standard VDG screen. If you are using windows, **ALT** lets you produce international characters. (See *OS-9 Windowing System Owner's Manual* for more information).
- The keyboard has an auto-repeat function. Holding down a key causes the character to repeat until you release the key. This function operates properly only when the disk drives are not in use by a program.
- You can deal with the video display and keyboard together as though they are a file. You can receive input from the keyboard and send output to the video screen using the device name `/TERM`.

Special Keys

The following keys and key sequences have special significance to OS-9.

ALT Produces graphic characters on a standard VDG screen or international characters with windows. Press **ALT** *char* (where *char* is a keyboard character).

CTRL A control key.

BREAK or **CTRL** **E** Stops the current program execution.

← or **CTRL** **H** Moves the cursor to the left one space.

CTRL _	Generates an underscore character.
CTRL {	Generates a left brace ({}).
CTRL }	Generates a right brace ({}).
CTRL ~	Generates a tilde (~).
CTRL \	Generates a backslash (\).
CTRL BREAK	Performs an ESCAPE function, and sends an end-of-file message to a program receiving keyboard input. To be recognized, CTRL BREAK must be the first thing typed on a line.
SHIFT BREAK or CTRL C	Performs a CONTROL C function by interrupting the video display of a program. The program runs as a background task.
CLEAR *	Selects the next video window.
SHIFT CLEAR *	Selects the previous video window.
	* You must have established windows for this function key to have any effect. See "Using Windows" in Chapter 7.
CTRL CLEAR	Toggles the <i>keyboard mouse</i> on and off. The keyboard mouse uses the arrow keys and the two function keys (F1 and F2) to simulate an external mouse. When keyboard mouse is on, the normal functions for the arrow and function keys is suspended.
SHIFT ← or CTRL X	Deletes the current line.
CTRL Q	Activates or deactivates the shift lock function.
CTRL 	Generates a vertical bar ().
CTRL ↑	Generates an up arrow (^).
CTRL [Generates a left bracket ([).

CTRL]

Generates a right bracket (]).

CTRL A

Redisplays the last line you typed and positions the cursor at the end of the line, but does not process the line. Press ENTER to process the line, or edit the line by backspacing. If you edit, press CTRL A again to display the edited line.

CTRL D

Redisplays the current command line.

CTRL W

Temporarily halts video output. Press any key to resume output.

ENTER

Performs a carriage return or executes the current command line.

OS-9 Toolkit

You now know about a number of OS-9 commands that can help you set up and use your computer system. There are many more commands available. This chapter contains information about a few of the most helpful commands. Becoming acquainted with these makes it easy for you to use other commands and functions. *OS-9 Commands* contains more information and a complete reference to all OS-9 commands (including those you have already discussed).

Viewing Directories

To look at your disk directories use the DIR command. For example, to view the contents of the current data directory, type:

```
dir 
```

If your data directory contains more filenames than can display on the screen at one time, the display pauses. Press the space bar to cause additional files to scroll onto the screen.

You can also view your execution directory in a similar manner. This time you must include the command option, x. Type:

```
dir x 
```

If you want to look at a directory on a disk drive other than the current drive, specify a complete path for OS-9 to follow, including the disk drive name. For example:

```
dir /d0/FAMILY/WORK 
```

Creating Directories

Before you can store data in a directory other than the ROOT directory, you must create that directory with MAKDIR. For instance, to create a FAMILY directory on your Drive /D0 diskette, type:

```
mkdir /d0/FAMILY 
```

Deleting Directories

You can also delete directories you create. **When you delete a directory you also delete any files or subdirectories it contains; so use this command with caution.** To delete a directory, follow these steps.

1. Use DIR to view the contents of the target directory and any of its subdirectories.
2. Copy any files you want to keep into a directory outside of the directory you want to delete.
3. Type:

```
deldir dirname 
```

where *dirname* is the name of the directory you want to delete.

The screen shows:

```
Deleting directory file.  
List directory, delete directory, or quit ?  
(l/d/q)
```

4. You now have three options:
 - a. To again confirm the contents of the directory before you delete it, press .
 - b. To initiate the deletion process, press .
 - c. To quit the process and leave the directory intact, press .

If you try to delete directories other than the ones you create, OS-9 might display Error #214, No Permission (you do not *own* the directory or have write permission for it). For information on handling such directories, see the ATTR command in *OS-9 Commands*.

Displaying Current Directories

There are times when you need to know the names of your current data and execution directories. The PWD and PXD commands make this possible. To determine your current data directory, type:

```
pwd 
```

The command displays the path from the ROOT directory to the current data directory. For instance, if your current data directory is PLEASURE (see Figure 4.5 in Chapter 4) the display is:

```
/D0/FAMILY/PLEASURE
```

To discover your current execution directory, type:

```
pxd 
```

The screen might display:

```
/D0/CMDS
```

A standard convention of OS-9 is to capitalize directory names. If you follow this convention when creating directories, you can always tell which files are directories at a glance.

Copying Files

COPY, like BACKUP, provides file security. If something happens to one file, you can use a copy. Also, you might want to copy a command or program to use in more than one directory, or you might want to use the same data on more than one computer.

Suppose you are in the PLEASURE directory of a diskette configured as in Figure 4.5. Your execution directory is the FAMILY directory, where you are using the Expenses program. Because the FAMILY directory does not contain any OS-9 commands, you have to change the execution directories whenever you want to use them.

You can make your work easier by copying the Expenses program to the CMDS directory. To do this, first make the CMDS directory your data directory by typing:

```
chd /d0/CMDS 
```

Then copy the Expenses file to the CMDS directory by typing:

```
copy /d0/FAMILY/expenses expenses 
```

Now, Expenses is in the CMDS directory, and you do not need to change the execution directory to FAMILY to use it.

Likewise, if the ROOT directory is your data directory, and you want to copy the Mom file from the WORK directory to the ROOT directory, type:

```
copy family/work/mom mom 
```

You can copy any file between directories and between disks. To do so, you must provide the COPY command with a pathlist for the location of the original file and for the destination of its copy.

Deleting Files

You can delete files in any directory using the DEL command, such as:

```
del myfile 
```

You can delete a file in the current execution directory by using the `-x` parameter. For instance, to delete Myprogram from the current execution directory, type:

```
del -x myprogram 
```

If the file you want to delete is in a directory other than the current data directory or the current execution directory, you must specify the full pathlist to the file. For instance, suppose you are in the ROOT directory of a diskette configured as Figure 4.5. To delete the Joe file in the WORK directory, type:

```
del family/work/joe 
```

If the file you want to delete is on a drive other than your current drive, include the drive name in your pathlist, such as:

```
del /d1/family/work/joe 
```

If you attempt to delete a file you did not create, OS-9 might display Error #214, No Permission. For information on deleting such files see the ATTR command in *OS-9 Commands*.

Renaming Files

OS-9 lets you change the names of files. Suppose Joe leaves home, and you now want to keep track of expenses for Sue. To change the name of the Joe file to Sue, type:

```
rename family/pleasure/joe sue 
```

Looking Inside Files

LIST is a command that lets you examine files that consist of text characters. For instance, to view the Dad file from the WORK directory, you might type:

```
list family/work/dad 
```

The contents of the file appears on the screen.

If you use LIST to display a file that is not a text file, it produces a meaningless display.

Loading Command Modules into Memory

When using OS-9, you might notice that some commands begin execution immediately, while others require access to the disk drive before they execute. The OS-9 commands you need most often load into memory at startup, so they are available for immediate use. If you plan to frequently use a command that is not in memory, you can *load* it.

For instance, the DSAVE command lets you copy an entire directory from one disk to another. To place the DSAVE module into your computer's memory, first be sure your execution directory is the CMDS directory, then type:

```
load dsave 
```

Now you can use DSAVE as many times as you want, without waiting for OS-9 to find it on disk.

Listing the Command Modules in Memory

At startup, OS-9 loads into memory the commands you use most often. If you are not sure whether a command already resides in memory, you can check using the MDIR command. To display a directory of the modules in your computer's memory, type:

```
mdir 
```

A list of all the modules in your computer's memory appear on the screen. The names you see are of modules OS-9 uses to boot and handle system operations and the commands it loads into memory when you boot the system.

Deleting Modules from Memory

After you load a module into memory, you can also delete it. The process is called *unlinking*. To delete the DSAVE command from memory, type:

```
unlink DSAVE 
```

Some modules might require unlinking more than once, depending on the number of times they were linked.

Do not attempt to unlink modules that you did not install in memory with the LOAD command.

Using Other Commands

OS-9 has nearly 50 commands and functions. This chapter has mentioned only a few. Not only are there other commands available through OS-9, several of the commands presented here have additional options.

The guidelines you learned in this handbook provide the background you need to make use of OS-9's many other capabilities.

By referring to *OS-9 Commands* you can learn how to create files, create *procedure files* to accomplish complicated tasks, send information to your printer, transfer data between devices, execute more than one task at the same time, and much more.

Customizing Your System

Your OS-9 operating system is originally configured in a certain way. For instance, it is set up to recognize two floppy disk drives, but no hard drives. It is set up to recognize a printer or one extra terminal. It does not recognize a modem. It assumes that you only want 32 characters across your computer's display screen. It provides all of the OS-9 commands.

Using the CONFIG utility from the BASIC09/CONFIG diskette that came with your OS-9 package, you can create system diskettes that match the computer system you have. Before proceeding further, be sure you have a working copy of the BASIC09/CONFIG diskette and a blank, formatted diskette. You can use the instructions in "Making Copies of Diskettes" in Chapter 3 to create a working copy of the BASIC09/CONFIG diskette and to create a blank, formatted diskette.

Creating a New System Diskette

To create a new system diskette **make sure you have a newly formatted diskette on hand**, then follow these steps:

1. Take out the System Master diskette, and replace it with the BASIC09/CONFIG diskette. Type:

```
chx /d0/cmds   
chd /d0   
config 
```

The first question the screen asks is:

```
HOW MANY DRIVES DO YOU HAVE:  
1 - ONE DRIVE ONLY  
2 - TWO OR MORE DRIVES  
SELECTION [1,2]
```

2. If you're using a single-drive system, press . If you have more than one drive, press .

If you indicated that you have two or more drives, CONFIG prompts:

```
ENTER NAME OF SOURCE DISK:
```

and

ENTER NAME OF DEST. DISK:

Type the appropriate drive name (/D0, /D1, etc.) at each prompt.

3. OS-9 informs you that it is:

```
BUILDING DESCRIPTOR LIST
.... PLEASE WAIT
```

OS-9 is putting together a list of the various devices you might want to use with your computer. When it finishes, it shows you the list:

```
                CONFIG
    ARROWS - UP/DOWN/MORE/BACK
    S - SEL/UNSEL H - HELP D - DONE
    ITEM                SEL
-----
    → P
      T1
      T2
      T3
      M1
      M2
      PIPE
      D0_35S           X
      D1_35S
      D2_35S
```

To view the rest of this menu, press **→**. Now the screen shows:

```

                CONFIG
      ARROWS - UP/DOWN/MORE/BACK
S - SEL/UNSEL H - HELP D - DONE
ITEM                SEL
-----
→  D3_35S
   DDD0_35S
   D0_40D
   D1_40D
   D2_40D
   DDD0_40D
   D1_80D
   D2_80D

```

4. You can choose the various devices you plan to use with your computer from this list. To choose a device, use **↑** or **↓** to move to the device. The **→** shows the device you've chosen. Then, press **S** (for Select) to display an X in the SEL ("Selected") column. Pressing **S** again cancels the selection.

You can move back and forth between the first and second screens by pressing either **→** (from the first screen) or **←** (from the second screen). Here's a short description of each device listed on this screen. To display helpful information about a device, position the **→** on its line in the list, and press **H** for Help. Then, press the space bar to make the help information disappear. The devices on this screen are:

- P A printer that connects to the RS-232 serial port on your computer.
- T1 A terminal using the standard RS-232 port (in addition to your main computer display).
- T2 A terminal using the optional RS-232 communications pak in Slot 1 of the Multi-pak Interface. T2 supports a full baud rate range. Use T2 in addition to your main computer display alone, or in addition to your main computer display and a "T1" type terminal.

T3	Another terminal using the optional RS-232 communications pak in Slot 2 of the Multi-pak Interface.
M1	A modem using an optional 300 baud modem pak.
M2	A modem using an optional 300 baud modem pak.
PIPE	Lets you use the PIPE utility in OS-9 (a utility that takes the information a program puts out and uses it as input data in another command).
D0_35S	Floppy Disk Drive /D0, single sided, 35 tracks.
D1_35S	Floppy Disk Drive /D1, single sided, 35 tracks.
D2_35S	Floppy Disk Drive /D2, single sided, 35 tracks.
D3_35S	Floppy Disk Drive /D3, single sided, 35 tracks.
DDD0_35S	Default Disk Drive /DD using Drive /D0, single sided, 35 tracks. Select one default drive — the drive where you keep your system diskette.
D0_40D	Floppy Disk Drive /D0, double sided, 40 cylinders.
D1_40D	Floppy Disk Drive /D1, double sided, 40 cylinders.
D2_40D	Floppy Disk Drive /D2, double sided, 40 cylinders.
DDD0_40D	Default Disk Drive /DD using Drive /D0, double sided, 40 cylinders. Select one default drive — the drive where you keep your system diskette.
D1_80D	Floppy Disk Drive /D1, double sided, 80 cylinders.
D2_80D	Floppy Disk Drive /D2, double sided, 80 cylinders.

You must select a "D0" device as your first disk drive—use D1, D2, and D3 devices for additional floppy disk drives. Select the drive that matches the drives you have on your system. If you are not sure, check with your supplier. To use extra terminals and modems, you must connect them via a Multi-Pak Interface.

5. As you finish choosing among the devices on the first screen, press to display another screen of devices:
6. When you finish selecting devices, press for Done. The screen asks:

```
ARE YOU SURE (Y/N) ?
```

7. Now's your chance to change your mind. Press if you want to reselect your devices. If you're sure about the devices you selected, press .

The next part of the CONFIG process appears on the screen:

```
CONFIG
SELECT TERM DESCRIPTOR
1 - TERM_VDG
2 - TERM_WIN
   H - HELP
SELECTION [1,2]
```

8. These are Color Computer terminal I/O subroutine modules you can use. For a 32 character display, select 1 (TERM_VDG). In order to have OS-9 windows and an 80 column display, select 2 (TERM_WIN).

Note: You can use TERM.WIN with a TV rather than a monitor but it is difficult, if not impossible, to see characters on an 80-column window. When you later create text windows, select 40-column displays.

If you select 2 (Term_Win), CONFIG presents you with another menu of choices. This time, the display looks like this:

```
                CONFIG
      ARROWS - UP/DOWN/MORE/BACK
S - SEL/UNSEL H - HELP D - DONE
      ITEM                SEL
-----
                W                X
                W1               X
                W2
                W3
                W4
                W5
                W6
                W7
```

This list represents the pre-established windows you can open for use with OS-9. The next section in this chapter tells you how to open and use windows. For now, if you expect to open windows in which you can run multiple tasks, select these items for your new diskette. (See "Using Windows" later in this Chapter.)

9. After you select the modules you want to use, press D. As it did when you selected devices, the screen asks ARE YOU SURE (Y/N) ? Press Y if you're finished. Or, press N to keep working on this screen.

OS-9 creates a file called Bootlist in Drive /D0's ROOT directory, using the information you've provided so far. It lets you know what it's up to by displaying:

```
BUILDING BOOT LIST
.... PLEASE WAIT
```

Then, the screen asks:

```
SELECT CLOCK MODULE:
  1 - 60 HZ (AMERICAN POWER)
  2 - 50 HZ (EUROPEAN POWER)
SELECTION [1,2]
```

10. Press 1 if you live in the United States, Canada, or any other country that uses 60Hz electrical power. If you live in a country that uses 50Hz electrical power, press 2.

11. CONFIG is ready to begin creating your customized System Master diskette. If you have one drive, the screen tells you that the DESTINATION diskette is your blank, formatted diskette and that your SOURCE diskette is the BASIC09/CONFIG diskette. Place your formatted diskette in the drive, and press **[C]**. You'll swap between the formatted diskette and the BASIC09/CONFIG diskette several times.

If you have a two-drive system, place a formatted diskette in Drive /D1, and press the space bar. The screen tells you that OS-9 is:

```
GENERATING NEW BOOT
.... PLEASE WAIT
```

12. Following the boot file generation, a menu lets you select the commands you want to include on your system diskette. You have the following choices: none; the full set of commands; or a set consisting of commands you choose individually. The menu looks like this:

```
CONFIG

DO YOU WISH TO ADD
[N]O COMMANDS, STOP NOW
[F]ULL COMMAND SET
[I]NDIVIDUALLY SELECT
[H] RECEIVE HELP
SELECTION [N,F,I,H]
```

Most people like to choose the individual commands they want to use. For the time being, press **[F]** to include the full set. Later, you can create another custom diskette that has only the commands you need.

13. Do one of the following:

- a. If you have one drive, the screen asks you to place your formatted diskette in Drive /D0. Do so, and press the space bar. Next, you'll place your "uncustomized" backup of the System Master diskette in Drive /D0. Swap the two diskettes as the screen asks you to. When the CONFIG program finishes, the 059: message reappears. You now have a brand new, customized copy of the System Master diskette.
- b. If you have more than one drive, the screen asks you to place your system diskette in Drive /D0. CONFIG continues and in a few minutes, finishes its work. The 059: message reappears, and you have a customized copy of the System Master diskette in Drive /D1.

14. Label the diskette so that you can distinguish between your working copy of the System Master diskette and the custom copy.

Monitor Types

OS-9 lets you set your system for different monitor types. The monitor options are for a RGB color monitor, a composite color monitor or TV, or a monochrome monitor or TV. To set your system for a particular monitor type, enter one of the following commands, or add it to your system Startup file:

Monitor Type	Command
RGB	montype r
Composite	montype c
Monochrome	montype m

Therefore, to set your system for a composite monitor, type:

```
montype c 
```

To save typing the command each time you start OS-9, put it in the Startup file in the ROOT directory of your system diskette.

If your system disk does not have an existing Startup file:

Create one by typing:

```
build startup   
montype r   

```

If your system disk already has a Startup file:

First rename the Startup file by typing:

```
rename startup oldstart 
```

Then create a file that contains the new command, such as:

```
build newstart   
montype r   

```

Now combine the two files into a new Startup file:

```
merge oldstart newstart > startup 
```

Use DEL to delete oldstart, newstart, or both, or leave them on your disk for future use.

Using Windows

If the window descriptors (W, W1, W2, W3, W4, W5, W6, W7) and the graphics interface and driver, GrfInt and GrfDrv, are in memory, OS-9 lets you set up windows on your display screen.

Note: GrfInt and the window descriptors must be loaded as part of the boot operation. Your System Master diskette does this.

Once you have initialized windows, you can then move among them, initiating different tasks in each. You can even have different processes showing on different portions of your display screen at the same time.

Another advantage of using windows is that you can choose windows that give you displays of 40 or 80 columns across the screen, rather than only 32. However, unless you have a monitor connected to your computer, rather than a television, you might be unable to read the screen.

Establishing a Window

You can establish one or more windows after booting OS-9, or you can include the window creation process in OS-9's Startup file. Startup is a file containing commands you want your system to execute during startup.

Getting Started With OS-9

To establish a window from the OS-9 prompt, type:

```
iniz wnumber   
shell i=/wnumber& 
```

In this example, *number* represents the window number to initialize. After you type these commands, you can select the window by pressing . To return to the original screen, press again.

The default values for the window descriptors /W1 through /W7 are:

Window device name	Text size in columns	Window's physical size	
		Starts at:	Size:
/W1	40	0,0	27,11
/W2	40	28,0	12,11
/W3	40	0,12	40,12
/W4	80	0,0	60,11
/W5	80	60,0	19,11
/W6	80	80,0	80,12
/W7	80	0,0	80,24

Note: To initialize Windows /W2 and /W3, you must be operating from Window /W1. To create Windows /W5 and /W6, you must be operating from Window /W4.

The "Starts at" column, indicates the position on the screen of the top left corner of the window. On the screen grid, coordinates 0,0 are located at the top left corner.

The "Size:" column indicates the number of characters across each window and the number of character lines in each window.

Therefore, Window 1 displays 40 column text, begins in the top left corner of the screen, extends right for 27 characters and down for 11 lines. Window 5 displays 80 column text, begins at the top of the screen, 60 columns from the left, extends 19 columns to the right and 11 lines down.

Note that the coordinates for each window are based on the text size of the screen. Therefore, Window 1 (based on 40 column text) ends at column 27, while Window 5 (based on 80 column text) begins at column 60.

Using the information in the previous chart, you can now establish any, or all, of the seven windows.

Note: You cannot establish all of the windows unless your computer has 512 kilobytes of memory.

For instance, to set up a full screen, 80-column window, type:

```
shell i=/w7& 
```

After a short pause, the screen displays a message, such as:

```
&004
```

This means that OS-9 has opened a path to your new window and started a shell on the window with the process identification of 04. To move to the window, press . Your 32-column screen vanishes and you are now in Window 7. You can type commands or run programs from here in the same manner as before.

To set up three windows on the same screen, type these commands, then use to move among the windows:

```
iniz w1 w2 w3   
shell i=/w1&   
shell i=/w2&   
shell i=/w3& 
```

If you want, and your computer has enough memory, you can run different processes in all of the windows.

Changing Window Colors

Perhaps you don't like the color of the screen in one or more of your windows. You can change it using the display command. The following charts show you all of the colors available for the screen background, text, and border.

Background Code = 33

Text Code = 32

Border Code = 34

Color Codes

<u>Codes</u>	<u>Color</u>
00 or 08	White
01 or 09	Blue
02 or 0A	Black
03 or 0B	Green
04 or 0C	Red
05 or 0D	Yellow
06 or 0E	Magenta
07 or 0F	Cyan

To change a color, type DISPLAY 1b, followed by the background, text, border, or foreground code followed by a color code. Then, press **ENTER**.

For instance, if you are in Window 7, you can change the background color to red, by typing:

```
display 1b 33 04 ENTER
```

Change the text color to black by typing:

```
display 1b 32 02 ENTER
```

To put a white border around the screen, type:

```
display 1b 34 00 ENTER
```

You can also type all the codes on one line, like this:

```
display 1b 33 04 1b 32 02 1b 34 00 ENTER
```

Pick the colors you want for each window, and change them using DISPLAY.

Eliminating a Window

In the command to establish windows (shell i=/wnumber&), "i" tells SHELL that the process being created is *immortal*. This means that you can only terminate it from the window in which it resides.

To kill a window in which you have established a shell, press **CLEAR** until the window you want appears on the screen. Type:

```
ex ENTER
```

Now press **CLEAR** to move to another window in which a shell is running. Then use **DEINIZ** to deinitialize that window. For instance, if the window you want to eliminate is Window 1, type:

```
deiniz w1 ENTER
```

Using Startup To Establish A Window

If you intend to use a window whenever you start OS-9, for instance if you want to use an 80 column screen, put the appropriate commands in the Startup file. This file must be located in the ROOT directory of your system disk.

If your system diskette already has a Startup file:

First rename the existing Startup file, such as:

```
rename startup oldstart ENTER
```

Then put your new commands into a temporary file. To initialize window Number 7 (80 columns, full screen) with white text on a black background, type:

```
build tempstart  
iniz w7 ENTER  
shell i=/w7& ENTER  
display 1b 32 00 1b 33 02 1b 34 02 0c > /w7 ENTER  
ENTER
```

Now combine your new commands with the original Startup file by typing:

```
merge oldstart tempstart > startup ENTER
```

You can remove the Tempstart file by typing `del tempstart ENTER`, or you can leave it in your ROOT directory for future use.

If Startup does not already exist:

Create it by typing:

```
build startup   
iniz w7   
display 1b 32 00 1b 33 02 1b 34 02 0c > /w7   
shell i=/w7&   

```

Now, after you boot OS-9, press to operate in an 80-column, black and white screen.

Index

- adding commands 7-7
- ALT 5-8
- anonymous directory names 4-6
- application 1-2
 - diskette 1-2
 - programs 1-2, 3-5
- arrow keys 1-2, 5-8, 5-9
- ASCII value 5-8
- auto-repeat, keyboard 5-8

- backslash character 5-9
- backup 3-5
 - diskettes 3-3
 - files 6-3
- BASIC09/CONFIG diskette 3-5, 3-7
- bit, high order 5-8
- booting OS-9 2-2
- brace 5-9
- bracket character 5-9, 5-10
- BUILD 7-8, 7-13

- care of diskettes 3-1
- carriage return 5-10
- changing
 - directories 5-6
 - the system diskette 5-7
- character
 - ASCII 5-8
 - backslash 5-9
 - brace 5-9
 - tilde 5-9
 - underscore 5-9
 - up arrow 5-9
 - valid 4-5
 - vertical bar 5-9
- clock module 7-6
- CMDS directory 4-1, 5-5
- colors, window 7-11, 7-12
- command 1-1, 6-1
 - accessing 5-4
 - adding 7-7
 - editing 5-1
 - loading 6-5
 - line, 1-1, 1-2, 5-1
 - using spaces 5-2
 - mistakes 5-1
 - modules, listing 6-5
 - option 5-2
 - parameters 5-2
 - process, 5-10
 - typing 5-1
- communications pak 1-3
- composite monitor 7-8
- computer, turning off 2-3, 3-1
- CONFIG 7-1
- configuring your system 7-1
- contents of directories 6-1
- control key 5-8
- CONTROL-C 5-9
- copies
 - with one drive 3-5
 - with two drives 3-7
- COPY 5-2, 5-4
- copying
 - diskettes 3-3, 3-5
 - files 6-3
- CTRL-BREAK 5-9
- CTRL-C 1-1
- CTRL-H 5-1
- CTRL-X 5-1
- current execution directory 5-5
- cursor, move 5-8
- customizing your system 7-1

- data 4-1
 - directory, changing 5-7
 - files 4-1
 - terminal 1-3
 - types 4-1
 - storing 4-1
- date 2-2
- DEINIZ 7-13
- deleting
 - lines 5-9
 - directory files 6-2
 - files 6-4
 - memory modules 6-6
- descriptors, window 7-10
- destination diskette 3-6
- device names 4-6, 7-3
- DIR 5-2

- directory 4-1
 - changing 5-6
 - changing the data 5-7
 - CMDS 4-1
 - contents 6-2
 - creating 6-1
 - current execution 5-5
 - deleting 6-2
 - display 6-2
 - finding 4-5
 - multiple 4-4
 - pathlist 4-5
 - ROOT 4-1, 4-5
 - SYS 4-2
 - viewing 6-1
- directory names 4-4
 - anonymous 4-6
 - displaying 6-2
- disk drive 1-4
 - names 3-2,
- disk name 3-3, 4-6
- diskette
 - backup 3-5
 - copying 3-3
 - formatting 3-3, 3-4
 - handling 3-1
 - removing 5-7
 - track 3-3
 - BASIC09/CONFIG 3-5
 - changing the system 5-7
 - destination 3-6
 - source 3-6
 - system 7-1.
 - write protect 3-2
- DISPLAY 7-12
- display
 - directory 6-2
 - directory names 6-2
 - file 5-2
 - video 5-8
- drive names 3-2
- DSAVE 6-5
- duplicate diskette 3-6, 3-7

- editing commands 5-1
- end-of-file 5-9
- entering commands 5-1
- error message 1-5, 2-4
- ESCAPE function 5-9
- examples of filenames 4-4

- execution directory 5-5
- exit OS-9 2-3
- extended DIR 5-3

- files 4-1
 - copying 6-3
 - deleting 6-4
 - display 5-2
 - finding 4-5
 - names 4-4
 - pathlist 4-5
 - renaming 6-4
 - Startup 7-8
 - viewing 6-5
- filename
 - examples 4-4
 - legal characters 4-4
 - finding 4-5
 - directories 4-5
- formatting
 - diskette 3-3, 3-4
 - with two drives 3-4

- graphics
 - characters 5-8
 - interface 7-9

- halt video output 5-10
- handling diskettes 3-1
- hard disk name 3-2
- hardware 1-3
- high order bit 5-8

- initialize diskette 3-3, 3-4
- INIZ 7-10
- interface, graphics 7-9
- international characters 5-8

- key, control 5-8
- keyboard
 - mouse 5-9
 - auto-repeat 5-8
- keys, arrow 5-8, 5-9
- kill a window 7-13

- languages 1-2
- last line, redisplay 5-10
- left brace 5-9
- left bracket 5-9
- legal characters, filenames 4-4

- length, command line 5-1
- line
 - command 5-1
 - delete 5-9
 - redisplay 5-10
- listing command modules 6-5
- loading commands 6-5
- lowercase letters 2-4, 5-1, 5-8

- memory modules 6-5
 - deleting 6-6
- menus 1-2
- messages, error 2-4
- mistakes, command 5-1
- modem 1-3, 1-4
- modifier 5-1
- module, clock 7-6
- modules
 - in memory 6-5
 - window 7-6
- monitor 5-8, 7-8
 - monochrome 7-8
- mouse, keyboard 5-9
- move cursor 5-8
- multi-pak interface 1-3, 2-1
- multi-tasking 1-4
- multi-user 1-4
- multiple directories 4-4

- names
 - anonymous 4-6
 - device 4-6
 - directory 4-4
 - disk drive 3-2
 - file 4-4
 - hard disk 3-2
 - of devices 7-3

- one drive copies 3-5
- operating system 1-1
- option, command 5-2
- OS-9, starting 2-1
- output, halt video 5-10

- parameter 5-1
 - command 5-2
- pathlist 4-5, 4-6
- periods, anonymous names 4-6
- peripherals 1-3
- process command 5-10

- program 4-1
 - application 3-5
 - execution, stopping 5-8
 - files 4-1
 - name 5-1
- protect diskettes 3-2

- quit OS-9 2-3

- rebooting OS-9 2-3
- redisplay
 - current line 5-10
 - last line 5-10
- removing
 - diskettes 5-7
 - windows 7-12
- renaming files 6-4
- reset button 2-3
- RGB monitor 7-8
- right brace 5-9
- ROOT directory 4-1, 4-2, 4-5
- route to files 4-5
- RS-232 1-3, 2-1

- scratched 3-6, 3-7
- screen, VDG 5-8
- sector 3-4
- select window 5-9
- serial port 1-3
- SHELL 7-11
- shift lock 5-9
- size of windows 7-10
- slash
 - in device names 4-6
 - in pathlist 4-5, 5-7
- SN error 2-2
- source diskette 3-6
- spaces in a command line 5-2
- special keys 5-8
- starting
 - OS-9 2-1
 - your computer 2-1
- Startup 7-8
 - file 7-13
- stopping program execution 5-8
- storing
 - data 4-1
 - diskettes 3-1
- subdirectory 4-2
- substitute names 4-6

- syntax error 2-2
- SYS directory 4-2
- system
 - customizing 7-1
 - devices 7-3
 - diskette 7-1
 - name 1-3
- TERM 5-8
- TERM_VDG 7-5
- TERM_WIN 7-5
- terminal 1-3
- text file 5-2
- tilde 5-9
- time 2-2
- track 3-3, 3-4
- turning off your computer 3-1
- two drive
 - copies 3-7
 - formatting 3-4
- types
 - of data 4-1
 - of monitors 7-8
- underscore character 5-9
- up arrow 5-9
- uppercase 2-4, 5-1, 5-8
- valid character 4-4, 4-5
- VDG screen 5-8
- vertical bar 5-9
- video
 - display 5-8
 - output, halt 5-10
 - window, select 5-9
- viewing files 6-5
- window 1-4, 7-9
 - 40-column 7-5
 - 80-column 7-5
 - colors 7-11, 7-12
 - descriptors 7-10
 - eliminating 7-12
 - establishing 7-9
 - modules 7-6
 - names 4-6
 - path 7-11
 - size 7-10
 - establish 7-13
- write protect 3-2, 3-3



OS-9

Commands

Reference

OS-9[™] Level Two Operating System
©1983, 1986 Microware Systems Corporation.
Licensed to Tandy Corporation.
All Rights Reserved.

OS-9 Commands:
©1986 Tandy Corporation
and Microware Systems Corporation.
All Rights Reserved.

Reproduction or use, without express written permission from Tandy Corporation or Microware Systems Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in preparation of this manual to assure its accuracy, neither Tandy Corporation nor Microware Systems Corporation assumes any liability resulting from any errors in or omissions from this manual, or from the use of the information contained herein.

Contents

Chapter 1 Introduction	1-1
The Kernel	1-1
The Input/Output Manager	1-2
Device Drivers	1-2
Device Descriptors	1-2
The Shell	1-3
Going On	1-3
Chapter 2 The OS-9 File System	2-1
Input/Output Paths	2-1
Disk Directories	2-2
Subdirectories	2-3
Disk Files	2-3
Sectors	2-4
Text Files	2-5
Random-Access Data Files	2-6
Procedure Files	2-6
Executable Program Module Files	2-7
Miscellaneous File Use	2-8
The File Security System	2-8
Examining and Changing File Attributes	2-9
Record Lockout	2-11
Device Names	2-12
Chapter 3 Advanced Features of the Shell	3-1
More About Command Line Processing	3-1
Command Modifiers	3-3
Execution Modifiers	3-3
Alternate Memory Size Modifier	3-3
I/O Redirection Modifiers	3-4
Command Separators	3-5
Sequential Execution Using the Semicolon	3-6
Concurrent Execution Using the Ampersand	3-6
Combining Sequential and Concurrent Executions	3-7
Using Pipes: the Exclamation Mark	3-7
Raw Disk Input/Output	3-8
Command Grouping	3-9

Shell Procedure Files	3-10
Built-in Shell Commands and Options	3-11
Running Compiled Intermediate Code Programs	3-12
Chapter 4 Multiprogramming and Memory Management	4-1
Processor Time Allocation and Timeslicing	4-1
Process States	4-2
Creation of Processes	4-3
Basic Memory Management Functions	4-5
Loading Program Modules Into Memory	4-6
Deleting Modules From Memory	4-7
Loading Multiple Programs	4-8
Chapter 5 Useful System Information and Functions	5-1
File Managers, Device Drivers, and Descriptors	5-1
The Sys Directory	5-2
The Startup File	5-3
The CMDS Directory	5-3
Making New System Diskettes	5-3
Technical Information for the RS-232 Port	5-4
Chapter 6 System Command Descriptions	6-1
Organization of Entries	6-1
Command Syntax Notations	6-1
Command Summary	6-3
Chapter 7 Macro Text Editor	7-1
Overview	7-1
Text Buffers	7-1
Edit Pointers	7-1
Entering Commands	7-2
Control Keys	7-2
Command Parameters	7-3
Numeric Parameters	7-3
String Parameters	7-4
Syntax Notation	7-4
Getting Started	7-4
Edit Commands	7-6
Displaying Text	7-6
Manipulating the Edit Pointer	7-7
Inserting and Deleting Lines	7-10

Searching and Substituting	7-13
Miscellaneous Commands	7-14
Manipulating Multiple Buffers	7-17
Text File Operations	7-18
Conditionals and Command Series Repetition	7-21
Edit Macros	7-25
Macro Headers	7-25
Using Macros	7-26
Macro Commands	7-28
Sample Session 1	7-32
Sample Session 2	7-38
Sample Session 3	7-40
Sample Session 4	7-45
Sample Session 5	7-49
Edit Quick Reference Summary	7-55
Edit Commands	7-55
Pseudo Macros	7-57
Editor Error Messages	7-59

Appendices

A OS-9 Error Codes	A-1
Device Driver Errors	A-5
B Color Computer 2 Compatibility	B-1
Alpha Mode Display	B-3
Using Alpha Mode Controls with Windows	B-3
Alpha Mode Command Codes	B-4
Graphics Mode Display	B-6
Graphics Mode Selection Codes	B-6
Graphics Mode Control Commands	B-7
Display Control Codes Summary	B-9
C OS-9 Keyboard Codes	C-1
D OS-9 Keyboard Control Functions	D-1

Index

Introduction

Getting Started With OS-9 contains the information you must know to use the system. However, the handbook reveals only a small part of OS-9's capabilities. To learn about all of its features, you need to know more about how OS-9 works. This introduction provides such basic background information.

The Kernel

At the center of the OS-9 system is a module (program) called a *kernel*. (See the following illustration.) The kernel provides basic system services, such as multitasking and memory management. It links other system modules and serves as the system administrator, supervisor, and resource manager.

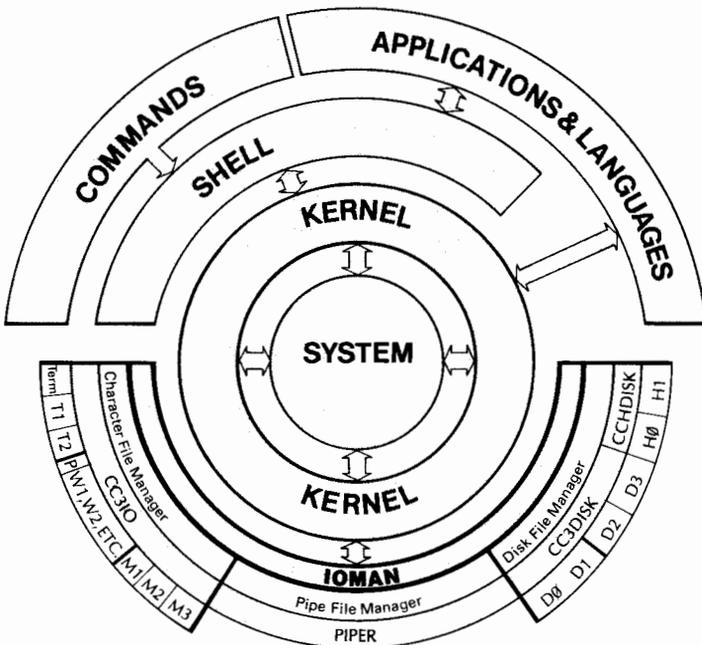


Figure 1

Term is your keyboard and video.
T1 and T2 are additional terminals.
P is a printer.
M1, M2, and M3 are modems.

The Input/Output Manager

Although the kernel manages OS-9, it does not directly process the input and output of data among the other modules and your computer *hardware* (printers, disk drives, terminals, and so on). Instead the kernel passes this responsibility to the input/output manager, IOMAN.

IOMAN has three *submanagers*: a character file manager, a pipe file manager, and a disk file manager. The responsibilities of these managers are as follows:

The Character File Manager Handles the transfer of data between OS-9 and *character devices* (devices that operate on a character-by-character basis, such as terminals, printers, or modems). The sequential character file manager (SCF) can handle any number or type of such devices.

The Pipe File Manager Handles communication between processes or tasks. Pipes let you use the output of one process as the input of another process.

The Disk File Manager This Random Block File Manager (RBF) handles the transfer of data to and from *block-oriented, random access* devices, such as a disk drive system.

Device Drivers

CC3IO, PIPER, and CC3DISK are *device drivers*. These files contain code that transforms standard data into a form acceptable to a particular device, whether it is a terminal, printer, modem, disk drive, any other device, or another file. PIPES transfers data between processes.

Device Descriptors

Term, T1, P, M1, D0, and so on, are *device descriptors*. These files describe the devices connected to the system. They contain device initialization data as well as code that directs OS-9 to the physical addresses of the ports to which devices are connected.

The Shell

The kernel, in conjunction with IOMAN and its associated managers and modules, make up the OS-9 operating system. These modules handle all of the system's functions. However, OS-9 needs directions before it can accomplish useful tasks.

Directions to the system have two sources: commands and applications or computer language programs.

Before commands are useful to the kernel, the *shell* must interpret them. It analyzes commands and converts them into code that the kernel can understand.

Some application programs and computer languages also use the shell's functions. Others can access the kernel directly and do not need to go through the shell.

Going On

Chapters 2 through 5 contain detailed information on the operation of the OS-9 system illustrated in Figure 1. These chapters more fully describe the composition of files and directories. They tell about advanced features of commands and of the shell and contain information on multiprogramming and memory management.

Chapter 6 contains descriptions of the OS-9 commands. Chapter 7 tells you how to use OS-9's Macro Text Editor.



The OS-9 File System

Input and output refer to the data your computer system receives and the data that it sends. OS-9 can receive (input) data from a keyboard, disk files, modems, and other terminals. It can send (output) data to all of these devices—except the keyboard—and to a video display.

OS-9 receives and sends data in the same format, regardless of whether the destination is a file or a device. It overcomes the differences in the devices by defining a standard for them and using *device drivers* to make each device conform to the standard. The result: a much simpler and more versatile input/output system.

Input/Output Paths

The base of OS-9's unified I/O system is an organization of *paths*. Input/output paths are, in effect, software links between files. (Remember, OS-9 thinks of all devices as files.)

Individual device drivers process data so that it conforms to the hardware requirements of the device involved. Data transfer is in streams of *8-bit bytes* that can be either bidirectional (read/write) or unidirectional (read only or write only), depending on the device, how you establish the path, or both. A byte is a unit of computer data. (A byte may contain the code for one alphabet character.) A bit is a binary digit and has a value of either 0 or 1.

OS-9 does not require the data it manages to have any special format or meaning. The meaning of the data is determined by the programs that use it.

Some of the advantages of such a unified I/O system are:

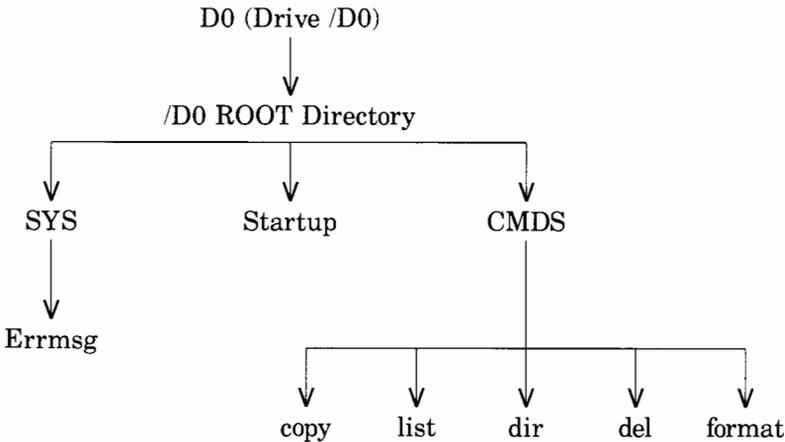
- Programs operate correctly regardless of the I/O devices selected.
- Programs are highly portable from one computer to another, even when the computers have different types of I/O devices.
- You can redirect I/O to alternate files or devices when you run a program, without having to alter the program.

- You can easily create and install new or special device driver routines.

Disk Directories

A directory is a storage place for other directories and files. It contains the information about the directories and files assigned to it so that OS-9 can easily find and access the data they contain.

Each disk has its own directory system. For example, a typical system diskette, diagrammed partially and simply, might look like this:



The *ROOT* directory of /D0—the *ROOT* from which the rest of the disk’s file system *grows*—contains a file called Startup and two directories, SYS and CMDS.

SYS and CMDS, in turn, contain files: SYS contains Errmsg, and CMDS contains Copy, List, Dir, Del, and Format. All these files and directories, and many more, come built into the OS-9 system.

OS-9 organizes each directory area into 32-byte *records*. The first 29 bytes contain filename characters. The first byte of the name has its *sign bit* (the leftmost or most significant bit) set. When you delete a file, it is not immediately destroyed. Rather, the deletion process sets the first character position of the record to zero, and OS-9 no longer recognizes the record. Although the file contents still exist, they are no longer accessible to you or OS-9. Subsequent file creations overwrite deleted records.

The last three bytes of a record make up a 24-bit binary number that is the *logical sector number* pointing to the file descriptor record. Logical sectors are numbered with reference to the sequence of their use, rather than their physical location on a disk. See “Disk Files” for more information on disk organization.

You create directories using the MAKDIR command and can identify them by the D (directory) attribute. (See “Examining and Changing File Attributes”.) MAKDIR initializes each directory with two entries having the names “.” and “..”. These entries contain the logical sector numbers of the directory and its parent directory, respectively.

You cannot use the COPY and LIST commands (as described in *Getting Started With OS-9*) with directories. Instead, use DSAVE and DIR.

You cannot delete directories directly. You must first empty a directory of files, convert it into a standard file, and then delete it. However, the DELDIR command performs all these functions automatically.

Subdirectories

A *subdirectory* is a directory residing in another directory. Actually, all directories you create are subdirectories, since all directories branch from the ROOT directory. However, because the system automatically creates the ROOT directory when you format a disk, this manual does not consider directories residing in the ROOT directory to be subdirectories.

Subdirectories can contain files and other subdirectories. In effect, OS-9 catalogues files and directories in much the same way that you might put files pertaining to a particular subject in a file cabinet drawer. With OS-9, you can have as many directory levels as your disk storage space permits.

Disk Files

A disk file is a *logical* block of data. (Logical means that although the data might not actually exist in a contiguous *block*, OS-9 treats it as though it does.) A file can contain a program, text, a command, a computer language, or any other form of code. Every time you ask OS-9 to store data on a disk, you must specify a *filename* for that block of data. Both you and the system must then use the filename to access the data.

The system stores all files as an ordered sequence of 8-bit bytes. The file can be any size from 0 bytes to the maximum capacity of the storage device and can be expanded or shortened as desired.

When OS-9 creates or opens a file, it establishes a *file pointer* for it. OS-9 addresses bytes within the file in the same manner it addresses memory, and the file pointer holds the *address* of the next byte to write or read. OS-9's *read* and *write* functions always update the pointer as the system transfers data.

This pointer function lets assembly-language programmers and high-level language programmers reposition the file pointer. To expand a file, write past the previous end of the file. Reading up to the last byte of a file causes the next read request to return an end-of-file status.

OS-9's file system also uses a universal organization for all I/O devices. This feature means that application programs can treat each hardware device similarly. The following section gives basic information about the physical file structure used by OS-9. (For more information, see the *OS-9 Level Two Technical Reference* manual.)

Sectors

The data contained in a file is stored in one or more *sectors* (disk storage units). These file sectors have both a *logical* and a *physical* arrangement. The logical arrangement numbers the sectors in sequence. The physical arrangement can be in any order based on the actual location of a sector on a disk's surface. For instance, Logical Sector 1 might be located at Physical Sector 10, and Logical Sector 2 might be located at Physical Sector 19.

Each sector contains 256 data bytes. The first sector of every file (Logical Sector Number 0 or LSN 0) is called the *file descriptor*. It contains the logical and physical description of the file. The disk driver module links sector numbers to physical track/sector numbers on a disk.

A sector is the smallest physical unit of a file that OS-9 can allocate for storage. On the Color Computer, a sector is also the smallest file unit. (To increase efficiency on some larger-capacity disk systems, OS-9 uses uniform-sized groups of sectors, called *clusters*, as the smallest allocatable unit. A cluster is always an integral power of two—2, 4, 8, and so on.)

OS-9 uses one or more sectors of each disk as a bitmap (usually starting at LSN 1) in which each data bit corresponds to one cluster on the disk. The system sets and clears bits to indicate which clusters it is using, which clusters are defective, and which clusters are free for allocation. The Color Computer default floppy disk system uses this format:

- Double-density recording on one side
- 35 tracks per diskette
- 18 sectors per track
- One sector per cluster

Each OS-9 file has a directory entry that includes the filename and the logical sector number of the file's *file descriptor sector*. The file descriptor sector contains a complete description of its file, including:

- Attributes
- Owner
- Date and time created
- Size
- Segment list (description of data sector blocks)

Unless the file size is 0, the file uses one or more sectors/clusters to store data. The system groups data sectors into one or more adjacent blocks called *segments*.

Text Files

Text files contain variable-length lines of ASCII characters. A carriage return (ASCII code 0D hexadecimal or 13 decimal) terminates each line. Text files contain such data as program source code, procedure files, messages, and documentation.

Programs usually read text files sequentially. Almost all high-level languages (such as BASIC09) support text files.

Use LIST to examine the content of text files.

Random-Access Data Files

Random-access files consist of sequences of records, with each record the same length. A program can find any record's beginning address by multiplying the record number by the number of bytes used for each record. This feature allows direct access of any record.

Usually, high-level languages let you subdivide records into fields. Each field can have a fixed length and use. For example, the first field of a record can be 25 text characters in length, the next field can be two bytes in length and used to hold 16-bit binary numbers, and so on.

OS-9 does not directly process records. It only provides the basic file functions used by high-level languages to create and handle random-access files.

Programmers use high-level languages like BASIC09, Pascal, and C to create random-access data files. For instance, in BASIC09 and Pascal, GET, PUT, and SEEK functions operate on random-access files.

Procedure Files

Procedure files are disk files that contain commands. You can use them to execute a series of commands by typing and entering a single command name.

Your System Master diskette contains one procedure file named Startup. You can create your own procedure files using the BUILD command, copying input from the keyboard to a file, or by using a text editor program. For instance, suppose you have three disk drives, /D0, /D1, and /H0. You could create three very simple procedures to allow you to look at the directories of these disks by typing and entering a simple two-character command.

To create a procedure file to look at the directory of /D1, type:

```
build p1 
display 0C 
dir /d1 
display 0A 

```

The first line creates a file named P1 (print directory for Drive /D1). When you press `ENTER`, a question mark appears on the screen telling you that OS-9 is waiting for input. Type the rest of the lines. Finally, press `ENTER` at the beginning of a line to tell OS-9 that the input is complete. OS-9 closes the file.

Now, to see the contents of Drive /D1, type `p1 ENTER`. The command `display 0C` clears the video screen. The command `display 0A` causes the cursor to drop down one line on the screen.

Use your imagination. Almost anything you can do from the keyboard, you can do with a procedure file. However, remember that OS-9 looks only in the current data directory for a procedure file, unless you provide a full pathlist to the procedure. Also, OS-9 must be able to find any command in the current execution directory that is part of a procedure file. If the current execution directory does not contain the commands you need, change it, either from the keyboard or as part of your procedure file.

Executable Program Module Files

OS-9 *program modules* are executable program code, generated by an assembler or compiled by a high-level language. A file can contain one or more program modules.

Each module has a standard format that includes the *object code* (the executable portion of the module), a *module header* that describes the type and size of the module, and a CRC (Cyclic Redundancy Checksum) value. The system stores program modules in files in the same structure in which they load into memory. Because OS-9 programs are position-independent, they do not require specific memory addresses for loading.

For OS-9 to load program module(s) from a file, the file `execute` attribute must be set, and each module must have a valid module header and CRC value. If you or the system alters a program module in any way (either as a file or in memory), its CRC check value becomes incorrect, and OS-9 cannot load the module.

If a file contains two or more modules, OS-9 treats them as a *group* and assigns contiguous memory locations for them.

Using LIST on program files or any other files that contain binary data, causes a jumbled display of random characters and an error message.

Miscellaneous File Use

OS-9's basic file functions are so versatile that you can devise almost unlimited numbers of special-purpose file formats for particular applications that require formats not discussed here (text, random-access, and so on).

The File Security System

Each file and directory has properties called *ownership* and *attributes* that determine who can access the file and how they can use it.

OS-9 automatically stores the user number associated with the creation of a file. The system considers the *owner* of the number to be the owner of the file.

Security functions are based on access attributes. There are eight attributes, each of which can be turned *off* or *on* independently. When the D (directory) attribute is on for a file, that file is a directory. (Only MAKDIR can set the D attribute for a file.) When the S (single-user) attribute is on, only one program or user can access the file at a time.

The other six attributes control whether the file can be read from, written to, or executed by either the owner or the *public* (all other users.) When on, these six attributes function as follows:

- Owner read permission** The owner can read from the file. Use this permission to prevent *binary* files from being used as *text* files.

- Owner write permission** The owner can write to the file or delete it. Use this permission to protect important files from accidental deletion or modification.

- Owner execute permission** The owner can load the file into memory and execute it. To be loaded, the file must contain one or more valid OS-9 memory modules.

- Public read permission** Anyone can read and copy the file.

- Public write permission** Anyone can write to or delete the file.

- Public execute permission** Anyone can execute the file.

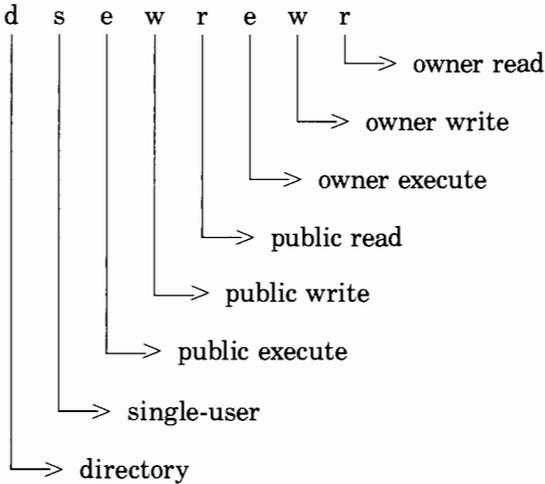
For example, if a file has all permissions on except *write permit to public* and *read permit to public*, the owner has unrestricted access to the file. Other users can execute it but cannot read, copy, delete, or alter it.

Examining and Changing File Attributes

You can use the DIR command, with the E (entire) option, to examine the security permissions of all files in a particular directory. An example of output using DIR E on the current data directory is:

```
Directory of . 10:20:44
Owner Last modified Attributes Sector Bytecount Name
-----
0 86/07/31 1436 ----r-wr A 6567 OS9Boot
0 86/07/31 1437 d-ewrewr 71 560 CMDS
0 86/07/31 1442 d-ewrewr 1B8 80 SYS
0 86/07/31 1409 -----wr 1Bd 55 startup
```

The *Attributes* column shows which attributes are on by listing one or more of the following codes.



For example, the first file shows:

```
----r-wr
```

This means that (1) The file is not a directory. (2) It is shareable. (3) The public cannot execute it or (4) write to it, but can (5) read it. (6) The owner cannot execute the file, but can (7) write to it, and (8) can read it.

To examine the attributes of a particular file, use ATTR. Typing ATTR followed by a filename shows you the file's current attributes, for example:

```
attr file2 
```

A possible screen display is:

```
---wr-wr
```

To change a file's attributes use ATTR and a filename, followed by a list of one or more attribute abbreviations. However, you must own a file before you can change its attributes.

The following command enables public write and public read permissions and removes the execute permission for both the owner and the public:

```
attr file2 pw pr -e -pe 
```

Note: In order to protect data stored in directories, the D attribute behaves somewhat differently from the other attributes. You cannot use ATTR to turn on the D attribute—only MAKDIR can do that—and you can use ATTR to turn off D only if the directory is empty.

Record Lockout

When two or more processes use the same file simultaneously, they might attempt to update the file at the same time, causing unpredictable results. When you open a file in the *update* mode, OS-9 eliminates the problem of simultaneous use by *locking* the sections of the file. The lock covers any disk sectors containing the bytes last read by each process accessing the file. If one process attempts to access a locked portion of a file, OS-9 puts the process to sleep until the locked area is free.

OS-9 moves the lock and frees the area when it reads from or writes to another area. The system removes a lock on a file when the process associated with the lock closes its path to the file. A process can have only one lock on a file, but it can have locks on more than one file.

You can lock an entire file by activating its single user bit. (See the earlier section “Examining and Changing File Attributes.”) When the single user bit is on, only one process can open a path to the file at a time. Attempts by other processes to access the file result in an error.

Device Names

Each physical I/O device supported by OS-9 has a unique name. The following list describes some of the device names supported by the system. Your system diskette already contains several of these devices. You can define others to use with CONFIG.

D0_35S	Floppy Disk Drive /D0, single sided, 35 cylinders.
D1_35S	Floppy Disk Drive /D1, single sided, 35 cylinders.
D2_35S	Floppy Disk Drive /D2, single sided, 35 cylinders.
D3_35S	Floppy Disk Drive /D3, single sided, 35 cylinders.
DDD0_35S	Default Disk Drive /D0, single sided, 35 cylinders.
D0_40D	Floppy Disk Drive /D0, double sided, 40 cylinders.
D1_40D	Floppy Disk Drive /D1, double sided, 40 cylinders.
D2_40D	Floppy Disk Drive /D2, double sided, 40 cylinders.
DDD0_40D	Default Disk Drive /D0, double sided, 40 cylinders.
D1_80D	Floppy Disk Drive /D1, double sided, 80 cylinders.
D2_80D	Floppy Disk Drive /D2, double sided, 80 cylinders.
P	a printer using the RS-232 serial port
TERM	your computer keyboard and video display
T1	a terminal port using the standard RS-232 port
T2	a terminal using the optional RS-232 communications pak
T3	a terminal using the optional RS-232 communications pak
M1	a modem using an optional 300 baud modem pak
M2	a modem using an optional 300 baud modem pak
W	a generic window descriptor
W1	window device Number 1

W2	window device Number 2
W3	window device Number 3
W4	window device Number 4
W5	window device Number 5
W6	window device Number 6
W7	window device Number 7

Although OS-9 and your computer can access all these devices, your original diskette does not configure it to do so. For information on configuring your system, see Chapter 7 of *Getting Started With OS-9*.

Because device names are at the root of the file system, you can use them only as the first part of a pathlist. Always precede the name of a device with a slash.

When you refer to a non-disk device, for example a terminal or printer, use only the device name. /P, for instance, is the full allowable pathlist for a printer.

Note: An I/O device name is actually the name of an OS-9 device descriptor that you precede with a slash (/). OS-9 automatically loads device descriptors during the OS-9 boot sequence. You can add or delete other device descriptors while the system is running or add device descriptors to the bootfile using CONFIG.



Advanced Features of the Shell

This chapter discusses the advanced capabilities of the shell. In addition to basic command line processing, the shell facilitates:

- Input/output redirection, including filters
- Memory allocation
- Multitasking (concurrent execution)
- Procedure file execution
- Built-in commands

You can use these advanced capabilities in many combinations. Following are several examples. Study the basic rules, use your imagination, and explore.

More About Command Line Processing

The shell is a program that reads and processes command lines, one at a time, from the computer's input device (usually your keyboard). It *parses* (scans) each line to identify and process any of the following parts that might be present:

- A program, procedure file, or built-in command
- Parameters to be passed to the program
- Execution modifiers to be processed by the shell

For some commands, only the *keyword* (the program, procedure file, or command name) must be present. Other commands have required or optional parameters. As well, a command line can include *modifiers* that influence the operation of the command. OS-9 features that affect command execution are:

Execution Modifiers Let you increase the amount of random access memory (RAM) available for a command. Also lets you redirect input to a process, output from a process, or both.

Command Separators Let you enter more than one command on a line, perform concurrent execution of commands, or connect the input or output of one command to another command.

Command Grouping Lets you group all the commands that you want affected by command modifiers or separators.

Note: The next section, "Command Modifiers," provides details on these features.

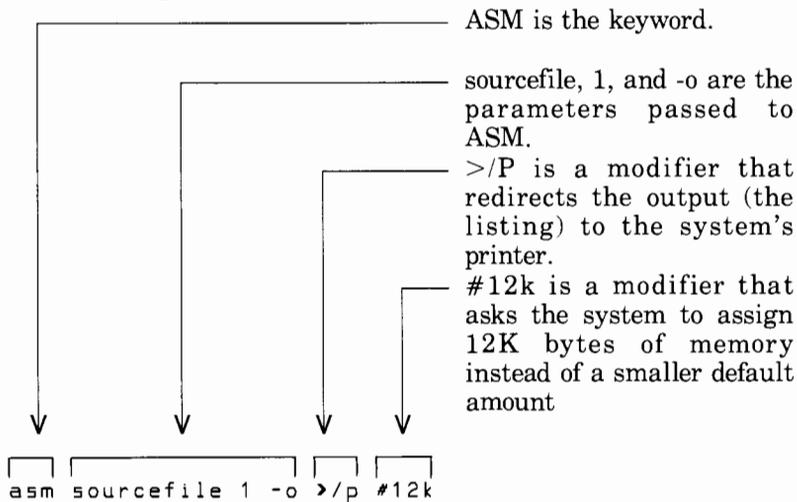
Once the shell identifies the keyword, it processes any modifiers. It then assumes the remaining text consists of parameters, which it passes to the program being called.

When the shell receives a built-in command, it immediately executes it. If it receives a command that is not built in, it searches for the appropriate program and then runs it as a new process. The keyword must be the first entry in any line.

While the program is running, the shell deactivates itself. At the termination of the program, the shell reactivates and accepts the next input. This cycle continues until the shell detects an end-of-file in the input path. It then terminates its own execution. You can input an end of file from the keyboard by pressing **SHIFT** **BREAK**.

Following is a sample shell command line that calls the assembler.

In this example:



Command Modifiers

Add command modifiers to a command line to change the way in which the command functions. Modifiers let you tailor OS-9 commands to your specifications. Type them in a command line after the keyword and either before or after other parameters you might be using.

The shell processes command modifiers before it executes a program. If it detects an error in any of the modifiers, it stops execution and reports the error.

The shell strips command modifiers from the part(s) of the command line passed to the program as parameters. Therefore, you cannot use the characters reserved as modifiers (# ; ! < > &) inside other parameters.

Execution Modifiers

Execution modifiers alter the amount of memory commands have available, or they redirect command input or output.

Alternate Memory Size Modifier. When the shell invokes a command program, it allocates the minimum amount of working RAM (random access memory) specified in the program's module header.

Note: All executable programs include a module header which holds the program's name, size, memory requirements, and other information. For information on viewing the contents of a module header, see the IDENT command.

You might want to increase a command's default memory size. You can assign memory either in 256-byte pages or in 1024-byte increments. To add memory in pages, use the modifier #*n*, where *n* is the number of pages. To add memory in 1024-byte increments, use the modifier #*n*K, where *n* is the number of 1024-byte increments.

The following two examples have identical results:

```
copy #8 file1 file2  (8 x 256 = 2048 bytes)
copy #2K file1 file2  (2 x 1024 = 2048 bytes)
```

I/O Redirection Modifiers. Input/output redirection modifiers reroute a program's I/O from the standard path to alternate files or devices.

One of OS-9's advantages is that its programs use standard I/O paths rather than individual, specific file, or device names. You can easily redirect the I/O to any file or device without altering the program itself.

Programs that normally receive input from a terminal or send output to a terminal use one or more of these three standard I/O paths:

- **Standard input path**—Routes data from the terminal's keyboard to programs. The standard input path is Path Number 0.

Use the less-than symbol (<) to redirect data to this path.

- **Standard output path**—Routes data from programs to the terminal's display. The standard output path is Path Number 1.

Use the greater-than symbol (>) to redirect data from this path.

- **Standard error output path**—Routes routine status messages (prompts and errors) to the terminal's display. (The name *error output path* is somewhat misleading, since many kinds of messages in addition to error messages travel the path.) The standard error path is Path Number 2.

Use two greater-than symbols (>>) to redirect data from this path.

When you use a redirection modifier in a command line, follow it immediately with a pathlist for the substitute device. For example, you can use LIST to redirect the contents of a file called Correspondence from the terminal to the printer, by typing:

```
list correspondence >/p ENTER
```

The shell automatically creates, opens, and closes files referenced by redirection modifiers as needed. **The system immediately restores normal I/O paths at the completion of any command using redirection modifiers.**

In the next example, the shell redirects DIR's output—a list of files in the MEMOS directory—to the file /D1/Savelisting:

```
dir /d0/memos >/d1/savelisting 
```

You can now view the contents of Savelisting by typing:

```
list /d1/savelisting 
```

OS-9 displays the file contents in a format similar to a directory listing.

```
Directory of /d0/memos
CMDS      SYS      startup
OS9Boot
```

You can use one or more redirection modifiers before the program's parameters, after the program's parameters, or both. However, use each modifier only once in a command.

The following example shows how you can use all of the redirection modifiers together to start BASIC09 on a device window and redirect all input and output to it:

```
basic09 <>>>/w1 
```

When you redirect multiple paths, you must use the redirection symbols in the proper order as shown here:

<u>Legal</u>	<u>Illegal</u>
<> /w1	>< /w1
<>> /w1	>>< /w1
>>> /w1	>>< /w1

Command Separators. You can include more than one command on a command line by using command separators. Command separators cause multiple commands to execute either sequentially or concurrently, depending on the separator you use.

Sequential execution means that one program must complete its function and terminate before the shell lets the next program execute. *Concurrent execution* means that two or more programs begin execution and run simultaneously.

Sequential Execution Using the Semicolon. Using a semicolon between commands on one line causes them to execute sequentially. For instance:

```
copy myfile /d1/newfile; dir >/p 
```

This command causes the shell to: (1) execute the COPY command, (2) enter a *waiting* state until COPY terminates, then awake, and (3) execute DIR.

If an error occurs in any program, the shell does not execute subsequent commands, regardless of the state of the SHELL command's X (stop on error) option.

Here are two more examples of commands using the semicolon:

```
copy oldfile newfile; del oldfile; list newfile  

```

```
dir /d1/myfile; list temp >/p; del temp 
```

Commands separated by semicolons are in fact separate and equal *child* processes of the shell.

Note: When one process creates another process, OS-9 calls the creator the *parent process* and the created process the *child process*. The child can become a parent by creating yet another process.

Concurrent Execution Using the Ampersand. You can use the ampersand (&) to cause multiple commands to run at the same time. Each command you specify runs as a separate *child* process of the shell. That is, for each process, the shell creates a separate shell to handle the operation. When the process is complete, the child shell terminates, or *dies*.

While more than one process is running, OS-9 divides execution time equally among the processes.

The number of programs that can run at the same time varies. It depends on the amount of free memory in the system and the memory requirements of the programs being executed.

An example of a simple command line using the & separator is:

```
dir >/p& 
```

The shell begins to run DIR, sending output to the printer. At the same time it displays both the number of the *forked* process (DIR) and a new prompt, like this:

```
&007
OS9:
```

To fork a process means to create a process as a branch of another process—a subroutine.

After using the ampersand to fork a background process, you can then enter another command, which the shell executes while output from your original command continues to go to the printer. This means you don't waste time waiting for OS-9 to finish a task. At times, the keyboard might not seem to respond to your typing, because characters do not appear on the screen. However, OS-9 stores the characters in the keyboard buffer and displays them as soon as the shell can accept input again.

If you have several processes running simultaneously and want information about them, use the PROCS command.

Combining Sequential and Concurrent Executions. You can, if you want, use both the concurrent and sequential command separators in one command line. For example:

```
dir >/p& list file1& copy file1 file2; del temp
ENTER
```

Because the & modifier joins the DIR, LIST, and COPY commands, these commands run concurrently. But, because a semicolon precedes the DEL command, DEL does not run until the other commands terminate.

Using Pipes: The Exclamation Mark. You can use the exclamation mark (!) to construct *pipelines* for OS-9 commands. Pipelines consist of two or more concurrently executing programs with standard input paths, and standard output paths or both, connected to each other with *pipes*.

Pipes are the primary means of transferring data from process to process. They are vital to interprocess communications. Pipes are first-in, first-out buffers, or *holding areas* for data.

The shell automatically buffers and synchronizes I/O transfers using pipes. A single pipe can direct data to several destinations or *readers*, and can receive data from several sources, or *writers* on a first-come, first-serve basis. An end-of-file occurs if a program attempts to read from a pipe when writers are not available to send data. Conversely, a write error occurs if a program attempts to write to a pipe when readers are not available.

Pipelines are created by the shell when it processes an input line with one or more pipe separators (!). For each pipe separator, the shell directs the standard output of the program on the left of the pipe separator to the standard input of the program on the right of the separator. The shell creates an individual pipe for each pipe separator in the command line. For example:

```
update <master_file ! sort ! write_report  
>/p 
```

This command redirects input from a path called `Master_file` to a file named `Update`. The output of `Update` becomes the input for the program `Sort`. The output of `Sort`, in turn, becomes the input for the program `Write_report`. Finally, the command redirects output from `Write_report` to the printer. The shell executes all programs in a pipeline concurrently. The pipes synchronize the programs so the output of one never *gets ahead* of the input request of the next program. This synchronization means that data cannot flow through a pipeline any faster than the slowest program can process it.

Raw Disk Input/Output. OS-9 has a special pathlist function to perform *raw* physical input/output operations on a disk. The pathlist consists of the device name, immediately followed by the "@" character.

This command causes OS-9 to treat the entire diskette in Drive /D0 as one logical file. The operation reads each byte of each sector, without regard to actual file structure. Commands such as `DIR`, `ATTR`, and `MFREE` use this feature to access sectors of disks that are not part of file data areas, such as header sectors.

Warning: When using this raw access, use extreme caution. Because you can write on any sector, you can easily damage file or directory structures and lose data. Using @ defeats any file security and record locking systems.

To convert a byte address to a logical sector number when using @, multiply the sector number by 256. Conversely, the logical sector number of a byte address is the byte address, modulo 256.

Command Grouping

You can enclose sections of command lines in parentheses to permit modifiers and separators to affect an entire set of programs. The shell processes the material in the parentheses by recursively calling itself to execute the enclosed command list.

For example, if you want to send directory listings of the ROOT directory of Drive /D0 and then the ROOT directory of Drive /D1 to the printer, you can type either:

```
dir /d0 >/p; dir /d1 >/p 
```

or:

```
(dir /d0; dir /d1) >/p 
```

The results are identical except that the system *keeps* the printer continuously in the second example. In the first example, another user could *steal* the printer between DIR commands.

You can group commands to cause programs to execute both sequentially and concurrently with respect to the shell that initiated them. For instance:

```
(del file1; del file2; del file3)& 
```

Here, the shell does the overall deleting process concurrently with whatever else you tell it to do, because you're using &. However, the shell deletes the three specified files sequentially because you're using semicolons within the parentheses.

Suppose you have a program named Makeuppercase that converts lowercase characters to uppercase and a program named Transmit that sends data to another computer, you can use a command line like this:

```
(dir cmds; dir sys) ! makeuppercase ! transmit  

```

The shell processes the output of the first DIR command and then the second. It sends all the DIR output to Makeuppercase, and Transmit sends all the output to another computer.

Shell Procedure Files

The shell is a *re-entrant program*. This means it can be simultaneously executed by more than one process. Like most other OS-9 programs, the shell uses standard I/O paths for routine input and output.

OS-9's shell offers you a special feature, a time and effort saver called a *procedure file*. A procedure file is a related group of commands, and when you run the file, you execute all the commands.

Other names for procedure file processing are *batch* and *background* processing. A procedure file becomes new input for the shell. By running a procedure file, you're using the shell to create a new shell, a *subshell* that accepts and carries out the commands in the procedure file.

Note: If you plan to use the same command sequences repeatedly, create a procedure file to do the job by using BUILD.

When you enter any command line, the shell looks for the specified program in memory or in the execution directory. If it cannot find the program there, it searches the data directory for a file with the specified name. If it finds the file, the shell automatically interprets it as a procedure file, and creates the subshell, which executes the commands listed in the procedure file.

Procedure files can also let the computer execute a lengthy series of programs while it is unattended, or even while it is running other programs.

There are two ways to run a procedure file. For instance, to execute a procedure file called Mailsequence, type either:

```
shell mailsequence [ENTER]
```

or

```
mailsequence [ENTER]
```

Both commands do the same thing: create a subshell to run the commands you've built into your Mailsequence procedure file.

To run a procedure file in a concurrent mode, use the ampersand (&) modifier. As long as memory is available, you can run any number of files concurrently.

You can even build procedure files to sequentially or concurrently execute other procedure files.

Note: If you use procedure files to run programs you don't intend to monitor closely, you can redirect standard output and standard error output to another file. Later you can review the file's contents. Output redirection eliminates the annoying output of shell messages on your terminal at random times.

Built-in Shell Commands and Options

The shell has a number of built-in commands and options. Whenever you use one of these functions, the shell executes it without loading it or creating a new process to execute it.

You can execute built-in functions alone, use them at the beginning of a command line, or use them following any program separator. You can separate adjacent built-in commands by spaces or commas.

The built-in commands and their functions are:

- CHD *pathlist*** Changes the data directory to the directory specified by the pathlist.
- CHX *pathlist*** Changes the execution directory to the directory specified by the pathlist.
- EX *modname*** Directly executes the module named. This function deletes the shell process so that it ceases to exist and executes the new module in its place. Use EX to replace the executing shell with the program specified by *modname*. You can also use EX without a module name to eliminate the current shell, for example, a shell you initialized in a window (see below).
- i = *devname*** Makes a shell an immortal shell. This means that when the shell ends, due to an EOF, OS-9 restarts it. Each time the shell restarts, it has the same data and execution directories. To kill an immortal shell, use EX to "chain" to a null process, such as:

ex

w	Waits for any process to terminate.
* text	Allows you to make a <i>comment</i> . The shell does not process text following the asterisk. Use this function to label operations in a procedure file.
kill procID	Stops the specified process.
setpr procID number	Changes the specified process's priority number.
x	Causes the shell to cease operation whenever an error occurs (a system default).
-x	Causes the shell to continue operation when an error occurs. Use this function in procedure files to enable the shell to continue to other commands if one command process fails because of a system error.
p	Turns the shell prompt and messages on (a system default).
-p	Inhibits the shell prompt and messages. Use this option in procedure files to disable screen display. Be sure to turn the prompt and message function back on afterward.
t	Makes the shell copy all input lines to output. Use this function with a procedure file to cause command lines to display as they execute.
-t	Sets the system so that it does not copy input lines to output (a system default).

Running Compiled Intermediate Code Programs

Before the shell executes a program, it checks the program module's language type. If it is not 6809 machine language, the shell calls the appropriate run-time system for that module.

For instance, if you have BASIC09 on your OS-9 system and want to run a BASIC09 I-code module called Adventure, you can type:

```
basic09 adventure 
```

or:

```
adventure 
```

or:

```
runb adventure 
```

In the last example, the shell uses the RUNB module to interpret the Adventure I-code module.



Multiprogramming and Memory Management

One of OS-9's most valuable capabilities is *multiprogramming*—sometimes called timesharing or multitasking. This feature lets your computer run more than one process at the same time. Multiprogramming can be a time saving advantage in many situations. For example, you can edit one program while the system prints another. Or you can use your Color Computer to control a household alarm system, lighting, and heating and at the same time use it for routine work or entertainment.

OS-9 uses multiprogramming regularly for internal functions. You can use it by putting an ampersand at the end of a command line. Doing this causes the shell to run your command as a *background*, or concurrent, task.

To run several processes simultaneously, OS-9 must coordinate its input/output system and CPU time and allocate its memory as needed. This chapter gives you some basic information about how OS-9 manages its resources to optimize system efficiency and make efficient multiprogramming a reality.

Processor Time Allocation and Timeslicing

CPU time is the most precious resource of a computer. If the CPU is busy with one task it cannot perform another. For example, many processes must wait for you to enter information from the terminal. While the process is waiting, your computer's CPU must also wait. Your computer is limited by your typing speed.

On many systems there is no way around such a bottle neck. However, OS-9 is more efficient. It assigns CPU time to processes only as they need it.

To do this, OS-9 uses *timeslicing*. Timeslicing, as described in the following paragraphs, lets all active processes share CPU time.

A real-time clock interrupts the Color Computer's CPU 60 times each second. The interruption points are called *ticks*, and the spaces between ticks are called timeslices.

OS-9 allocates timeslices to each process. At any tick it can suspend execution of one process and begin execution of another. This starting and stopping of processes does not affect their execution.

How often OS-9 gives a process timeslices depends on the process's priority relative to the priority of other active processes. You can access priority using a decimal number from 0 through 255, where 255 is the highest priority.

OS-9 automatically gives the shell a priority of 128. Because child processes inherit their parents' priorities, the shell's child processes also have priorities of 128. You can find a process's priority with the `PROCS` command, and can change it using the `SETPR` command.

You cannot compute the exact percentage of CPU time assigned to any particular process, because there are some dynamic variables involved, such as the time the process spends waiting for I/O devices. But you can approximate the percentage by dividing the process's priority by the sum of the priority of all active processes:

$$\text{process's CPU share} = \frac{\text{priority of the process}}{\text{sum of the priorities of all active processes.}}$$

Note: Timeslicing happens so quickly that it looks as if all processes execute simultaneously and continuously. If, however, the computer becomes overloaded with processing, you might notice a delay in response to input from the terminal. Or, you might notice that a procedure program takes longer than usual to run.

Process States

The CPU time allocation system automatically assigns each process one of three *states* that describes its current status. Process states are important for coordinating process execution. A process can have only one state at any instant, although state changes can be frequent. The states are:

- **Active**—Applies to processes currently able to work—that is, those not waiting for input or for anything else. These are the only processes assigned CPU time.

- **Waiting**—Applies to processes that the system suspends until another process terminates. This state allows coordination of sequential process execution. The shell, for example, is in the waiting state during the execution of a command it has initiated.
- **Sleeping**—Applies to a process suspending itself for a specified time, or until receipt of a *signal*. (Signals are internal messages that coordinate concurrent processes.) This is the typical state of processes waiting for input/output operations.

The shell does not assign CPU time to sleeping or waiting processes. It waits until they become active. The PROCS command gives information about process states.

Creation of Processes

If a parent process creates more than one child process, the children are called *siblings* with respect to each other. If you examine the parent/child relationship of all processes in the system, a hierarchical lineage becomes evident. In fact, this hierarchy resembles a family tree. (The *family* concept makes it easy to describe relationships between processes.) OS-9 literature uses the family concept extensively in describing OS-9's multiprogramming functions.)

OS-9's *fork* function automatically performs the sequence of operations required to create a new process and initially allocate resources to it.

If for any reason, *fork* cannot perform any part of the sequence, the system stops and *fork* sends its parent an error code. The most frequent reason for failure is the unavailability of required resources (especially memory), or the inability of the system to find the specified program.

A process can create many processes, subject only to the availability of unassigned memory. When the parent issues a *fork* request to OS-9, it must specify certain information:

- **A primary module**—The name of the program to be executed by the new process. The program can already be present in memory, or OS-9 can load it from a disk file with the same name.

- **Parameters**—Data to be passed to and used by the new process. OS-9 copies this data to part of the child process's memory area. (Parameters frequently pass file-names, initialization values, and other information.)

The new process *inherits* some of its parent's properties, including:

- **A user number**—For use by the file security system to identify all processes belonging to a specific user. (This is not the same as the *process ID*, which identifies a process.) OS-9 obtains this number from the system password file when a user logs on. The system manager is always User 0.
- **Standard input and output paths**—The three paths: input, output, and error, used for routine input and output. Most paths can be shared simultaneously by two or more processes.
- **Current directories**—The data directory and the execution directory.
- **Process priority.**

As part of the fork operation, OS-9 automatically assigns:

- A process ID, a number in the range 1 to 255 that identifies the process. Each process has a unique number.
- Enough memory to support the new process. In OS-9, all processes share a memory address. OS-9 allocates a data area for the process's parameters and variables and a stack for each process's use. It needs a second memory area in which to load the process if it does not reside in memory.

In summary, each new process has:

- A primary module
- Parameters
- A user number
- Standard I/O paths
- Current directories
- A priority
- An ID number
- Memory

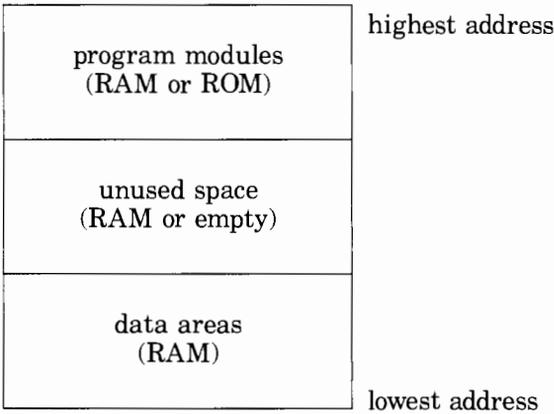
Basic Memory Management Functions

Memory management is an important OS-9 function. OS-9 automatically allocates all system memory to itself and to processes, and also keeps track of the logical contents of memory (the program modules that are resident in memory at any given time). The result is that you seldom need to bother with the actual memory addresses of programs or data.

The operating system and each process have individual address spaces. Each address space has the potential to contain up to 64 kilobytes of RAM memory. Using memory management unit (MMU) hardware, OS-9 moves memory into and out of each address space as required for system operations.

Although each unit is subject to the 64K maximum program size, you can run several processes simultaneously and utilize more than 64K overall. The system logically assigns RAM memory in 256-byte pages, but the MMU's hardware block size is 8K. Each of these physical blocks has an extended address that is called a *block number*. For example, the 8K physical block residing at address \$3C000 to \$3DFFF is Block Number \$3C.

Within an address space, OS-9 assigns memory from higher addresses downward for program modules and from lower addresses upward for data areas. The following chart shows this organization:



Loading Program Modules into Memory

When performing a fork operation, OS-9 first attempts to locate the requested program module by searching the *module directory*, which has the address of every module present in memory. The 6809 instruction set supports a type of program called *re-entrant code*, which means that processes can share the code of a program simultaneously.

Since almost all OS-9 family software is re-entrant, the system can make the most efficient use of memory. For example, suppose that OS-9 receives a request (from a process) to run BASIC09 (which requires 22 kilobytes of memory), but has already loaded it into memory for another process. Because the software is re-entrant, OS-9 does not have to load it again and use another 22K of memory. Instead the new process shares the original BASIC09 by including the location of the BASIC09 module in its memory map.

OS-9 automatically keeps track of how many processes are using each program module, and deletes the module when all processes using it terminate.

If the requested program does not yet reside in memory, OS-9 uses its name as a pathlist (filename) and attempts to load the program from disk.

Every program module has a module header describing the program and its memory requirements. OS-9 uses the header to determine how much memory the process needs for variable storage. The module header includes other information about its program, and is an essential part of the OS-9 machine language operation.

You can also place commands or programs into memory using the LOAD command. Doing so makes the program available to OS-9 at any time, without having to be loaded from disk. A program is physically loaded into memory only if it does not already reside there.

LOAD causes OS-9 to copy the requested module from a file into memory, verifying the CRC (Cyclic Redundancy Check). If the module is not already in the module directory, OS-9 adds it.

If the program module is already in memory, the load process still begins in the same way. But, when OS-9 attempts to add the module to the module directory and notices that the module is already there, it merely increments the known module's *link count* (the number of processes using the module).

When OS-9 loads multiple modules in a single file, it associates them logically in the memory management system. You cannot deallocate any of the group modules until all modules have zero link counts. Similarly, linking to one module within a group causes all other modules in the group to be mapped into the process's address space.

Deleting Modules From Memory

UNLINK is the opposite of LOAD. It decreases a program module's link count by one. When the count becomes zero (presuming that the module is no longer used by any process), OS-9 deletes the module, deallocates its memory, and removes its name from the module directory.

Warning: Never use the UNLINK command on a program or a module not previously installed using LOAD. Unlinking a module you did not LOAD (or LINK) might permanently delete it when the program terminates. The shell automatically unlinks programs loaded by fork.

Suppose you plan to use the COPY command ten times in a row. Normally, the shell must load COPY each time you enter the command. But if you load the COPY module into memory and then enter your string of commands, you don't have to wait for the system to load and unload COPY repeatedly. When you finish using COPY, use UNLINK to unlock the module from memory. The sequence looks like this:

```
load copy   
copy file1 file1a   
copy file2 file2a   
copy file3 file3a   
copy file4 file4a   
copy file5 file5a   
copy file6 file6a   
copy file7 file7a   
copy file8 file8a   
copy file9 file9a   
copy file10 file10a   
unlink copy 
```

It is important to use UNLINK when you no longer need the program. Otherwise, the program continues to occupy memory that might be used for other purposes.

Warning: Be careful not to unlink modules that are in use, because OS-9 deallocates the memory used by the module and destroys its contents. All programs using the unlinked module crash.

Loading Multiple Programs

Because all OS-9 program modules are *position-independent*, you can have more than one program in memory at the same time. Since position-independent code (PIC) programs don't have to be loaded into specific, predetermined memory addresses to work correctly, you can load them at different memory addresses at different times.

PIC programs require special types of machine language instructions that few computers have. The ability of the 6809 microprocessor to use PIC programs is a powerful feature and one of the greatest aids toward multiprogramming. You can load any number of program modules until available system memory is full.

OS-9 automatically loads each program module at non-overlapping addresses. (Most operating systems write over the previous program's memory when loading a new program.) OS-9's technique means that you do not need to be concerned with absolute memory addresses.



Useful System Information and Functions

The OS-9 system must load many parts of the operating system during startup and system operation. Therefore, on a floppy disk system, you must keep the system diskette in Drive /D0.

Two files used during the system startup operation, OS9Boot and Startup, must remain in the system diskette's ROOT directory. Other files on the system diskette are organized into two directories: CMDS (commands) and SYS (other system files). You can also create other files and directories on the system diskette. OS-9 always creates the initial data directory, or ROOT directory, when you format a diskette.

File Managers, Device Drivers, and Descriptors

The *bootstrap* (instructions that initialize OS-9) loads a file called OS9Boot into RAM memory at startup. This file contains file managers, device drivers and descriptors, and any other modules that permanently reside in memory. For instance, the OS9Boot file might contain these modules:

OS9p2	OS-9 Kernel
INIT	System Initialization Table
IOMan	OS-9 input/output manager
RBF	Random block (disk) file manager
SCF	Sequential character (terminal) file manager
PipeMan	Pipeline file manager
Piper	Pipeline driver
Pipe	Pipeline device descriptor
CC3IO	Keyboard/video graphics device driver
VDGINT	32x16 screen subroutines
GRFINT	Windowing subroutines
PRINTER	Printer device driver
SIO	RS-232 serial port device driver
CC3Disk	Disk driver
D0, D1	Disk device descriptor
TERM	Terminal device descriptor
T1	RS-232 serial port device descriptor
P	Printer (serial) device descriptor
P1	Printer (serial) device descriptor

Clock	Real-time clock module
CC3GO	System startup process
W - W7	Window device descriptors W, W1, W2, W3, W4, W5, W6, W7

OS-9 stores the modules loaded during the system startup with a minimum of fragmentation. To include additional modules, create new bootstrap files using the OS9GEN command or the CONFIG program supplied with OS-9. You cannot unlink a module loaded as part of the bootstrap.

After booting, when the system switches the boot block into its own address space, any non-system files included in the bootstrap decrease the memory available in the system mode. It is best to place optional modules in a separate file and load them as part of the system startup procedure. One example is the shell. Never include the shell as part of a system boot file in OS-9 Level Two systems.

The Sys Directory

The OS-9 SYS directory contains a number of important files:

- Errmsg is the error message file.
- Helpmsg contains syntax and usage information.
- Stdfonts contains the standard software fonts for use on graphic windows.
- Stdpats_2, Stdpats_4, and Stdpats_16 contain screen background and fill patterns for 2, 4, and 16 color graphics screens, respectively.
- Stdptrs contains graphic pointer images for use with a mouse.

These files, and the SYS directory itself, are not required to boot OS-9, but you do need them if you plan to use the ERROR or HELP commands, or if you intend to use text, or mouse pointers on graphic windows. You can also add other system-wide files of a similar nature.

The Startup File

The Startup file (/D0/startup) is a shell procedure file that OS-9 automatically processes as part of the system boot. You can include any legal shell command line in the Startup file. Many people include SETIME to start the system clock. If this file is not present, the system starts correctly, but the system time is not accurate.

The CMDS Directory

The directory /D0/CMDS is the system-wide command directory normally shared by all users as their working execution directory. The shell resides in the CMDS directory. The system start-up process CC3go makes CMDS the initial execution directory. You can add your own programs to the CMDS directory and have them execute in the same manner as the original system commands.

Making New System Diskettes

Getting Started With OS-9 told you how to create new system diskettes using the CONFIG utility. There are other ways to create system diskettes and either add or subtract capabilities. The following information provides guidelines on how to do this. For more detailed instructions see the descriptions of the CONFIG, OS9GEN, and COBBLER commands in this manual.

Before starting any of the following procedures, you need a blank, formatted diskette on which to place your system files. Then, choose one of the following methods to update your system:

- Use the OS9GEN command to add modules to the existing OS9Boot file.
- Use CONFIG to select the modules you want to include in the OS9Boot file.

If you choose to use CONFIG, the utility creates a complete system during the process. If you use OS9GEN, follow these steps:

1. Create the OS9Boot file using OS9GEN.
2. Create or copy the Startup file.
3. Copy the CMDS and SYS directories and the files they contain.

You can perform these steps manually or do them automatically by using one of these methods:

- Creating and using a shell procedure file
- Using a shell procedure file generated by DSAVE

Technical Information for the RS-232 Port

You can operate the RS-232 port or the printer at all standard baud rates from 110 baud to 19200 baud. (The default rate is 9600 baud for /t2, and 600 baud for /p.) The default format used is 8 data bits, no parity, and 1 stop bit.

Use the XMODE command to set the port's baud rate, parity, word length, stop bits, end-of-line delay, auto line feed, and so forth. To examine the printer's current settings, type:

```
xmode /p 
```

Then, if you want to make changes, use XMODE with information from the following chart. Select the parameter you want from the left column of each chart, and then select the corresponding number from the "Value to Use" column and write it down. After you select the proper value from each chart, add them together to obtain a final value for XMODE. All values must be hexadecimal.

Stop Bits		Word Length		Baud Rate	
Number of Stop Bits	Value to Use	Word Length	Value to Use	Bits Per Second	Value to Use
1 Stop Bit	0	7 Bits	20	110 BPS	0
2 Stop Bits	80	8 Bits	0	300 BPS	1
				600 BPS	2
				1200 BPS	3
				2400 BPS	4
				4800 BPS	5
				9600 BPS	6
				19200 BPS	7

For instance, to set the printer parameters to one stop bit, a word length of seven bits, and a baud rate of 600, select 0 from the Stop Bits chart, 20 from the Word Length chart, and 2 from the Baud Rate chart. Add the values together:

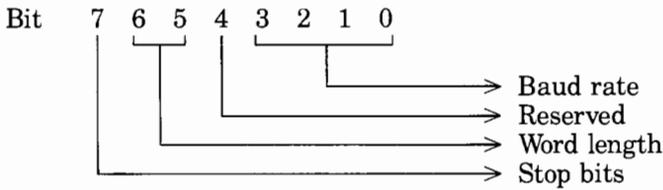
$$0 + 20 + 2 = 22$$

The command to set the printer port for this configuration is:

```
xmode /p baud=22 
```

When you use XMODE to set baud, parity, and stop bit values, you are actually setting the bits of a special byte to certain values. OS-9 uses these values to determine how to handle subsequent input/output operations. A bit is a binary digit and can be either 1 or 0. A byte consists of eight bits and can represent a value between 0 and 255.

The following chart shows the bits that control baud rate, word length, and stop bits for input/output operations on a specified device.



If the stop bit value = 0, stop bits = 1
 If the stop bit value = 1, stop bits = 2

If the word length value = 00, word length = 8 bits
 If the word length value = 01, word length = 7 bits

If the baud rate value = 0, baud rate = 110
 If the baud rate value = 1, baud rate = 300
 If the baud rate value = 2, baud rate = 600
 If the baud rate value = 3, baud rate = 1200
 If the baud rate value = 4, baud rate = 2400
 If the baud rate value = 5, baud rate = 4800
 If the baud rate value = 6, baud rate = 9600
 If the baud rate value = 7, baud rate = 19200
 (/t2 ACIAPAK only)
 If the baud rate value = 7, baud rate = 32000
 (/t1 SIO only)

Use `XMODE TYPE=value` to set parity, MDM (modem) kill, and the not ready delay. *Value* is a hexadecimal value you calculate from the following chart:

Parity

MDM Kill

Not Ready Delay

Type of Parity	Value to Use
None	0
Mark	A0
Space	E0
Even	60
Odd	20

Kill Switch	Value to Use
On	10
Off	0

Not Ready Delay	Value to Use
0 seconds	0
1 second	1
2 seconds	2
3 seconds	3
↓	↓
↓	↓
↓	↓
15 seconds	F

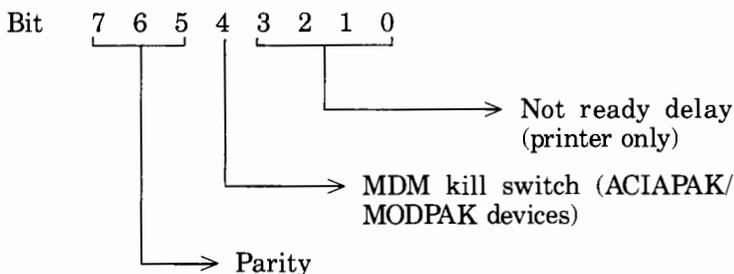
Select a value from each chart, and add them together to get a final TYPE value. For instance, to select even parity, MDM kill off, and a not ready delay of 10 seconds, select these values and add them:

$$60 + 0 + A = 6A$$

To set the new values, type:

```
xmode /p type=6a 
```

The following chart shows the bits that control parity, the modem kill switch, and the not ready delay.



If the parity value is 000, then parity = none

If the parity value is 101, then parity = MARK, no check

If the parity value is 111, then parity = SPACE, no check

If the MDM kill switch value is 0, then DCD loss = no kill

If the MDM kill switch value is 1, then DCD loss = kill

The value of the not ready delay bits equals the number of seconds delay.

For more information on setting other parameters, such as the end-of-line delay (null count), see the XMODE command reference in Chapter 6.



System Command Descriptions

This chapter contains alphabetical descriptions of the commands supplied with OS-9. Ordinarily, you call the commands from the shell, but you can also call them from most other programs in the OS-9 family—including BASIC09 and the Macro Text Editor.

Warning: Do not attempt to use OS-9 Level One commands with the OS-9 Level Two system or to use Level Two commands with the Level One system.

Organization of Entries

Each command entry includes:

- The name of the command
- A *syntax* line, which shows you the format and spelling to use when you type the command
- A brief definition of what the command does
- Information about any options available with the command
- Notes about the command and how to use it
- One or more examples of command use

Command Syntax Notations

OS-9 requires that you enter the various parts of a command in the correct order and in the correct format. An example of the proper *syntax* follows the command name.

The syntax line always begins with the name of the command. Occasionally, that's all you need (except for pressing `ENTER`). But other commands either require, or can accept, parameters (variables that give instructions to OS-9).

Some syntaxes include *variables* (shown in italics) that you replace with specific parameters. For instance, the BUILD command syntax is:

```
build filename ENTER
```

BUILD is the command name. You type it exactly as shown. However, *filename* is a variable. Replace it with the actual name you want to give to the file you are creating. If you want to create a file named Myfile, type:

```
build myfile 
```

Pressing executes the command.

Common variables are:

<i>arglist</i>	arglist (argument list) is similar to <i>paramlist</i> , but it includes command names as well as command parameters.
<i>devname</i>	device name (/P, /TERM, /M1)
<i>commandname</i>	command name
<i>dirname</i>	directory name
<i>filename</i>	file name
<i>hex</i>	a hexadecimal number
<i>hh/mm/ss</i>	hour/minutes/seconds
<i>modname</i>	name of a memory module
<i>n</i>	a decimal number
<i>number</i>	a numeric value
<i>opts</i>	options
<i>paramlist</i>	a list of parameters
<i>pathlist</i>	a complete path to a directory or file
<i>permission</i>	file permission abbreviations
<i>procID</i>	process ID number
<i>text</i>	a string of characters
<i>tickcount</i>	a numeric value representing system clock ticks
<i>value</i>	a numeric value
<i>yy/mm/dd</i>	year/month/day

[] Brackets indicate that the material within them is optional and not necessary for the execution of the command.

... An ellipsis indicates that you can repeat the material immediately preceding the ellipsis. For instance, [*filename*][...] means that you can specify more than one filename to the command. Following is the syntax for the DISPLAY command:

```
display hex [...] 
```

This means you can include more than one hex number with DISPLAY, such as:

```
display 54 48 49 53 20 49 53 20 41 20 53 45 43  
52 45 54 20 4D 45 53 53 41 47 45 
```

Command syntaxes do not include the shell's built-in options (for instance I/O redirection) because the shell filters out its options before it passes the command line to the program being called.

Command Summary

This section describes the format and use of OS-9 commands.

The following list is a summary of these commands:

- ATTR** Changes a file's attributes
- BACKUP** Makes a copy of a diskette
- BUILD** Builds a text file
- CHD** Changes the working data directory
- CHX** Changes the working execution directory
- CMP** Compares files
- COBBLER** ... Makes an OS9Boot file
- CONFIG** Creates a system diskette to your specifications
- COPY** Copies data
- DATE** Displays the system date and (optionally) the time
- DCHECK** Checks a disk file structure
- DEINIZ** Deinitializes a device previously initialized with INIZ
- DEL** Deletes a file or files
- DELDIR** Deletes a directory's files, then deletes the directory
- DIR** Displays the names of all files in a directory
- DISPLAY** Displays the characters represented by hexadecimal values
- DSAVE** Generates a procedure file to copy files

ECHO Echoes text to the screen
EDIT Calls the OS-9 Macro Text Editor
ERROR Displays a description of the last error code
EX Causes the shell process to execute another process
FORMAT Prepares a disk for data storage
FREE Displays the amount of free space on a disk
HELP Displays the syntax and use of commands
IDENT Displays OS-9 module identification
INIZ Initializes and attaches devices
KILL Terminates a process
LINK Links a module into memory
LIST Lists the contents of disk data files
LOAD Loads a module into memory
MAKDIR Creates a directory
MDIR Displays the names of the modules in memory
MERGE Copies and combines files
MFREE Displays a list of free RAM
MODPATCH.. Makes changes to a module in memory
MONTYPE ... Establishes the type of monitor in use
OS9GEN Builds and links a bootstrap file
PROCS Displays the names of the current processes
PWD Displays the name of the current data directory
PXD Displays the name of the current execution directory
RENAME Changes the name of a file or directory
SETIME Activates and sets the system clock
SETPR Sets a process's priority
SHELL Creates a child shell to process one or more commands
TMODE Changes the terminal's operating mode
TUNEPOR ... Adjusts the loop delay for the baud rate of /P or /T1 devices
UNLINK Unlinks memory modules
WCREATE ... Creates a window
XMODE Displays or changes a device's initialization parameters

ATTR

Syntax: `attr filename [permission]`

Function: Lets you examine or change a file's security permissions.

Parameters:

filename The name of the file you want to examine or change.

permission One or more of the following attribute options.

Options:

The file permission abbreviations you can use are:

- d Changes a file directory file to not a non-directory file.
- s Specifies that the file is not single-user and can serve only one user at a time.
- r Specifies that only the owner can read the file.
- w Specifies that only the owner can write to (change) the file.
- e Specifies that only the owner can execute the file.
- pr Specifies that the public (anyone) can read the file.
- pw Specifies that the public (anyone) can write to the file.
- pe Specifies that the public (anyone) can execute the file.
- a Tells ATTR not to display the attributes. Use this option when you wish to change attributes without displaying them.

Notes:

- To use ATTR, type the command name followed by the name of the file you want to change. Next, type a list of the permissions to turn on or off. Turn a permission on by typing its abbreviation or off by typing its abbreviation preceded by a minus sign. ATTR has no effect on permissions you do not name.
- If you do not specify any permissions, ATTR displays the file's current attributes.
- You cannot change the attributes of a file you don't own. User 0 can change the attributes of any file in the system.
- Use ATTR to change a directory into a file after deleting all the directory's files. You cannot change a file to a directory with ATTR. (See MAKDIR.)

Examples:

- To remove public read and write permission from a file named Myfile, type:

```
attr myfile -pr -pw 
```

- To give read, write, and execute permission to everyone for the file Myfile, type:

```
attr myfile r w e pr pw pe 
```

- To display the current permissions of a file named Datalog, type:

```
attr datalog 
```

BACKUP

Syntax: backup [*opts*][*devname1*][*devname2*]

Function: Copies all data from one disk to another.

Parameters:

<i>devname1</i>	The drive containing the disk files you want to back up.
<i>devname2</i>	The drive containing the disk to which you want to transfer the files.
<i>opts</i>	One or more of the following options.

Options:

e	Cancels the backup if a read error occurs.
s	Lets you backup a diskette using only one drive.
-v	Tells BACKUP not to verify the data written to the destination diskette.
# <i>n</i> K	Increases to <i>n</i> the amount of memory that BACKUP can use. Increasing the amount of memory assigned to BACKUP speeds the procedure. <i>n</i> can be either in pages of 256 bytes or in kilobytes (1024 bytes). Include K to indicate kilobytes.

Notes:

- BACKUP performs a sector by sector copy, ignoring file structures. In all cases, the devices specified *must* have the same format (size, density, and so forth) and the destination disk must **not** have defective sectors.

- If you omit both device names, the system assumes you are copying from /D0 to /D1. If you omit only the second device name, OS-9 performs a single-drive backup on the specified drive.
- The following demonstrates a complete backup of /D0 to /D1. In the example, the diskette in Drive /D1 is a formatted diskette with the name MYDISK. *Scratched*, which appears in one of the following messages, means erased. You type:

```
backup 
```

The screen display and your input are:

```
Ready to backup from /d0 to /d1 ? :   
MYDISK  
  is being scratched  
OK? :   
Sectors copied: $0276  
Verify pass  
Sectors verified: $0276
```

- Following is an example of a single-drive back up. BACKUP reads a portion of the source diskette (the diskette you are copying) into memory. It then prompts you to remove the source diskette and put the destination diskette (the diskette receiving the copy) into the drive.

After BACKUP writes to the destination diskette, remove the destination diskette and put the source diskette back into the drive. Continue swapping as prompted until BACKUP copies the entire diskette.

Giving BACKUP as much memory as possible means you have to make fewer diskette exchanges. If enough free memory is available, you can assign up to 56 kilobytes for the backup operation. An Error 207 means that your computer does not have the specified amount of memory free. To assign 32 kilobytes to backup, type:

```
backup /d0 #32k 
```

The screen display and your responses are as follows:

```
Ready to backup from /d0 to d0 ? :  Y
Ready Destination, hit a key:  ENTER
MYDISK
  is being scratched
OK?:  Y
Ready Source, hit a key:  ENTER
Ready Destination, hit a key:  ENTER
Ready Source, hit a key:  ENTER
Ready Destination, hit a key:  ENTER
```



```
Ready Destination, hit a key:  ENTER
Sectors copied: $0276
Verify pass
Sectors verified: $0276
```

In this procedure, the dollar symbol (\$) indicates hexadecimal numbers. BACKUP copied 276 hexadecimal (or 630 decimal) sectors.

Examples:

- To back up the diskette in Drive /D2 to the diskette in Drive /D3, type:

```
backup /d2 /d3  ENTER
```

- To back up from Drive /D0 to Drive /D1, without verification, type:

```
backup -v  ENTER
```

BUILD

Syntax: `build filename`

Function: Builds a text file by copying input from the standard input device (the keyboard) into the file specified by *filename*.

Parameters:

filename The name of the file you are creating.

Notes:

- BUILD creates a file, naming it *filename*. It then displays a question mark (?) and waits for you to type a line. When you type a line and press `ENTER`, BUILD writes the line to the disk.
- When you finish entering the lines for the new file, press `ENTER`, without any preceding text, to close the file and terminate the operation.
- The following example demonstrates how to build a text file named Newfile:

```
build newfile ENTER
? THE POWERS OF THE OS-9 ENTER
? OPERATING SYSTEM ARE TRULY ENTER
? FANTASTIC. ENTER
? ENTER
```

- To view the newly created file, type:

```
list newfile ENTER
```

The screen displays:

```
THE POWERS OF THE OS-9  
OPERATING SYSTEM ARE TRULY  
FANTASTIC.
```

Examples:

- To create a new file called `Small_file` and put into it whatever you type at the keyboard, type:

```
build small_file 
```

- To direct whatever you type to the printer, type:

```
build /p 
```

- You can use `BUILD` to transfer, or redirect, data from one file to another. Instead of the keyboard, this example uses a file named `Mytext` file for the input device. The output device is Terminal 1.

```
build <mytext /t1 
```

CHD CHX

Syntax: **chd** *pathlist*
 chx *pathlist*

Function: CHD changes the current working (data) directory, and CHX changes the current execution directory.

Parameters:

pathlist Specifies the directory for the current working or execution directory.

Notes:

- CHD and CHX do not appear in the CMDS directory because they are built into the shell.

Examples:

- To change the current working (data) directory to the PROGRAMS data directory located on the diskette in Drive /D1, type:

```
chd /d1/programs 
```

- To change the execution directory to the parent directory of the current execution directory, type:

```
chx .. 
```

- To change the execution directory to TEXT_PROGRAMS, a subdirectory of BINARY_FILES, type:

```
chx binary_files/text_programs 
```

- To return the execution directory and the data directory back to the default directories, type:

```
chx /d0/cmds; chd /d0 
```

Or, if you are using a hard disk, type:

```
chx /h0/cmds; chd /h0 
```

CMP

Syntax: `cmp filename1 filename2`

Function: Opens two files and compares the binary values of corresponding data bytes in the files. If CMP encounters any differences in the file, it displays the file offset (address) and the values of the bytes from each file.

Parameters:

filename1 are the files to compare.
filename2

Notes:

- The comparison ends when CMP encounters an end-of-file marker in either file. CMP then displays a summary of the number of bytes compared and the number of differences found.

Examples:

- To compare two files named Red and Blue, type:

```
cmp red blue 
```

Following is a sample screen display:

```
Differences
byte      #1  #2
-----  --  --
00000013  00  01
00000022  B0  B1
0000002A  9B  AB
0000002B  3B  36
0000002C  6D  65

Bytes compared: 0000002D
Bytes different: 00000005
```

- To compare two files that are identical, such as Red1 and Red2, type:

```
cmp red1 red2 
```

The screen display might be:

```
Differences
```

```
None ...
```

```
Bytes compared: 0000002D
```

```
Bytes different: 00000000
```

COBBLER

Syntax: `cobbler devname`

Function: Creates the OS9Boot file required on any OS-9 boot diskette.

Parameters:

devname The disk drive containing the diskette on which you want to create a new OS9Boot file.

Notes:

- COBBLER creates the new OS9Boot file with the *same* modules loaded during the most recent bootstrap. (To add modules to the bootstrap file, use OS9GEN.) COBBLER also writes the OS-9 kernel on Track 34 and excludes these sectors from the diskette allocation map. If any files are present on these sectors, COBBLER displays an error message.
- The new boot file must be contiguous on the diskette. For this reason, you should use COBBLER only with a newly formatted diskette. If you use COBBLER on a diskette that does not have a storage block large enough to hold the boot file, COBBLER destroys the old boot file, and OS-9 cannot boot from that diskette.
- To change device attributes permanently, use XMODE before using COBBLER.

Examples:

- To save the attributes of the current device on the system diskette, type:

```
cobbler /d0 
```

If you use COBBLER on a diskette that is not newly formatted, the screen displays:

```
WARNING - FILE(S) OR KERNEL  
PRESENT ON TRACK 34 - THIS  
TRACK NOT REWRITTEN
```

CONFIG

Syntax: config

Function: Lets you create a system diskette that includes only the device drivers and commands you select. CONFIG automatically adjusts its screen display for either 32- or 80-column display.

Notes:

- When executed, CONFIG displays menus of all I/O options and system commands. You select only those options and commands you want to include on a new system diskette.

Creating such a system diskette lets you make the most efficient use of computer memory and system diskette storage.

- The CONFIG utility is on the BASIC09/CONFIG diskette. Copy this diskette, using the OS-9 BACKUP command. Make the copy your working diskette. Keep the original in a safe place to use for future backups. After you boot your system, you can put the working copy of the BASIC09/CONFIG diskette in drive /d0. Then, type these commands:

```
chx /d0/cmds; chd /d0/modules 
```

- CONFIG does not require initial parameters. You establish parameters during the operation of the command. Be sure the execution directory is /D0/CMDS before executing CONFIG.
- You could save time by using BACKUP to create a system disk, using CONFIG to create a new boot file, and then deleting unwanted commands. However, this process causes fragmentation of diskette space and results in slower disk access. CONFIG causes no fragmentation.

- The MODULES directory of the BASIC09/CONFIG diskette contains all the device drivers and device descriptors supported by OS-9. The filename extension describes the type of file, as noted in the following table:

<u>Extension</u>	<u>Module Type</u>
.dd	Device Descriptor module
.dr	Device Driver module
.io	Input/Output subroutine module
.hp	Help file
.dw	Window Device Descriptor module
.dt	Terminal Device Descriptor module
.mn	File Manager module

Examples:

The following steps take you through the complete CONFIG process:

1. With the BOOT/CONFIG diskette in the current drive, type:

```
config 
```

2. CONFIG asks whether you want to use one or two disk drives. Press for single- or for two-drive operation.

If you specify one drive, continue with Step 3.

If you specify two drives, a display asks you to:

```
ENTER NAME OF SOURCE DISK:
```

```
Type /d0 
```

A display now asks you to:

```
ENTER NAME OF DEST. DISK:
```

```
Type /d1 
```

3. After a pause to build a descriptor list, the program displays a list of the various devices from the MODULES directory. Use and to move to a device. To include the device on the system diskette, press once. CONFIG displays an X by the selected device. To exclude a selected device, press again to erase the X.

A special help command provides information about each device. To display information about the current device (the device indicated by the arrow (→)), press **[H]**.

The list of devices might require more than one screen. Use **[→]** to move ahead page by page and **[←]** to move back.

The devices you can select and their descriptions are listed in Chapter 2 under the section "Device Names."

You must select a "D0" device as your first disk drive. Select from the list of D devices for other floppy disk drives. Select P to use a printer with OS-9, T1 to use a terminal, M1 to use a modem, and so on.

4. After selecting the devices you desire, press **[D]**. The screen displays, ARE YOU SURE (Y/N) ? If you are satisfied with your selections, press **[Y]**. If you want to make changes, press **[N]**.
5. To use your computer keyboard and video display, you must select either TERM_VDG or TERM_WIN. You use TERM_VDG for a 32-column display. For a TERM window that enables you to select character displays up to 80-columns, select TERM_WIN.
6. CONFIG builds a boot list from the selected devices and their associated drivers and managers in the MODULES directory of the current drive. It next displays two clock options:
 - 1 - 60HZ (AMERICAN POWER)
 - 2 - 50HZ (EUROPEAN POWER)
7. If you live in the United States, Canada, or any other country with 60hz electrical power, press **[1]**. If you live in a country with 50hz power, press **[2]**.

If you have a single disk drive, a screen prompt asks you to swap diskettes and press **[C]**. When asked for the SOURCE diskette, insert the BASIC09/CONFIG diskette. When asked for the DESTINATION diskette, insert the diskette that is to be your new OS-9 system diskette.

If you have more than one drive, a screen prompt asks you to insert a blank formatted diskette (the DESTINATION diskette) in the destination drive. The rest of the boot file creation is automatic.

8. After creating the boot file, CONFIG displays a menu of the commands you can include on your system diskette. You have the following choices:

[N] No Commands, Stop Now	— Do not add any commands
[F] Full Command Set	— Add all OS-9 commands from the current CMDS directory
[I] Individually Select	— Select commands one by one
[H] Receive Help	— Get help on the command set

Press **[N]** if you want to transfer a new boot file to a diskette on which you have previously copied the OS-9 system. If you have only one disk drive, this procedure is quicker than using the CONFIG utility to complete the entire system transfer, because it requires fewer disk swaps.

Press **[F]** to make an exact copy of the CMDS directory on your source diskette with a new boot file.

Press **[I]** to individually select commands to copy on the new diskette. The **[I]** option displays a menu similar to the device selection screen. Press **[S]** to select or exclude commands, and use the arrow keys to move among the commands in the menu. CONFIG selects files marked with an X for inclusion on the new system diskette. If a command does not have an X beside it, CONFIG excludes it from the new system diskette.

9. If you have a multi-drive system, a prompt appears asking you to insert your OS-9 system diskette in the destination drive. Press the space bar. The process finishes the CONFIG operation, and returns to OS-9.

If you have a single-drive system, you swap diskettes during the final process. This time, the SOURCE diskette is the OS-9 System diskette. The DESTINATION diskette is the system diskette you are creating. The number of swaps depends on the number of options you select.

Note: When using CONFIG you do not have to use your system diskette as the source diskette to install the commands. The program can use any diskette that contains a CMDS directory.

COPY

Syntax: `copy pathlist1 pathlist2 [opts]`

Function: Copies data from one file or device to another file or device.

Parameters:

- pathlist1* The name of the existing file or device from which you want to copy.
- pathlist2* The name of the device or file to receive the copy. If you are copying data to a file, the file must not already exist.

Options:

- s Causes COPY to perform a single-drive copy operation. *pathlist2* must be a full pathlist if you use -s. In a single-drive procedure, COPY reads a portion of the source disk into memory and then asks you to exchange the source and the destination diskette and press [C]. COPY might ask you to exchange diskettes several times before it completes duplicating the entire file.
- #n[K] Allows the use of more memory for the COPY procedure. If you specify K, *n* represents the amount of memory you want to use, in units of 1024 bytes. If you do not specify K, *n* represents the number of 256-byte memory pages. Using this option can increase speed and reduce the number of diskette swaps required for single-drive copies.

Notes:

- If *pathlist2* is a disk file, COPY automatically creates it. Data can be of any type, and COPY does not modify the file in any way.
- COPY does not add important codes (for example, line feeds). Use LIST instead of COPY when sending a text file to a terminal or printer.
- Following is an example of the screen display and your responses for COPY using a single drive:

```
copy /d0/cat /d0/animals/cat -s #32k 
Ready DESTINATION, hit C to continue: 
Ready SOURCE, hit C to continue: 
Ready DESTINATION, hit C to continue: 
```

↓

↓

This example assigns 32 kilobytes of memory for COPY to use. If enough free memory is available, you can specify up to 56 kilobytes. Copy continues asking you to swap the source and destination diskettes until the transfer is complete.

Examples:

- To copy File1 to File2 using 15K of memory, type:

```
copy file1 file2 #15k 
```

- To copy the News file on the diskette in Drive /D1 to a new file named Messages on the diskette in Drive /D0, type:

```
copy /d1/joe/news /d0/peter/messages 
```

DATE

Syntax: `date [t]`

Function: Displays the current date.

Options:

`t` Causes the time to appear with the date.

Notes:

- Following is an example of how to use SETIME to set a new date and time for the system and how to use DATE to check system date and time:

```
setime 
```

A possible screen display and your responses follows:

```
yy/mm/dd hh.mm.ss  
Time? 86/08/22 14.19.00 
```

```
date 
```

```
August 22, 1986
```

```
date t 
```

```
August 22, 1986 14.20.20
```

Examples:

- To display the system date and time, type:

```
date t 
```

- To direct the DATE command's output to the printer, type:

```
date t >/p 
```

DCHECK

Syntax: `dcheck [-opts] devname`

Function: Checks a disk's file structure.

Parameters:

devname The disk drive to check.
opts One or more of the following options.

Options:

-s Counts the number of directories and files and displays the results. This option causes DCHECK to check only the file descriptors for accuracy.
-b Suppresses listing of unused clusters (clusters allocated but not in the file structure).
-p Prints pathlists for questionable clusters.
-w = *pathlist* Specifies a path to a directory for work files.
-m Saves allocation map work files.
-o Prints DCHECK's valid options.

Notes:

- Sometimes the system allocates sectors on a disk that are not actually associated with a file or with the disk's free space. This situation can happen if you remove a disk from a drive while files are open. You can use DCHECK to detect this condition, as well as check the general integrity of directory/file links.

- After verifying and printing some vital file structure parameters, DCHECK follows pointers down the disk's file system tree to all directories and files on the disk. As it does so, it verifies the integrity of the file descriptor sectors, reports any discrepancies in the directory/file links, and builds a sector allocation map from the segment list associated with each file. If any file descriptor sectors (FDS) describe a segment with a cluster not within the file structure of the disk, DCHECK displays a message like this:

```
*** Bad FD segment ($xxxxxx-$yyyyyy) for file:
    (pathlist)
```

This message indicates that a segment starting at sector `xxxxxx` and ending at sector `yyyyyy` is not on the disk. If any of the file descriptor sectors are bad, the entire FD might be defective. DCHECK does not update the allocation map for corrupt FDS.

- While building the allocation map, DCHECK also ensures that each disk cluster appears once and only once in the file structure. If it discovers duplication, DCHECK displays a message like this:

```
Cluster $xxxxxx was previously allocated
```

This message indicates that DCHECK has found cluster `xxxxxx` more than once in the file structure. DCHECK reprints the message each time a cluster appears in more than one file.

Then, DCHECK compares the newly created allocation map with the allocation map stored on the disk and reports any differences with messages like these:

```
Cluster $xxxxxx in allocation map but not in file
    structure
```

```
Cluster $xxxxxx in file structure but not in
    allocation map
```

The first message indicates that sector number `xxxxxx` (hexadecimal) is not part of the file system, but the disk's allocation map has assigned it. FORMAT might exclude some sectors from the allocation map because they are defective.

The second message indicates that the cluster starting at sector *xxxxxx* is part of the file structure, but the disk's allocation map has not assigned it. Later operations might allocate this cluster, overwriting the contents of the cluster with data from the newly allocated file. (Clusters that DCHECK previously allocated can have this problem.)

- DCHECK builds its disk allocation map in a file called *pathlist/DCHECKpp0*, where *pathlist* is specified by the *-w* option and *pp* is the process number in hexadecimal. Each bit in this bitmap file corresponds to a cluster of sectors on the disk. If you use the *-p* option, DCHECK creates a second bitmap file (*pathlist2/DCHECKpp1*) that has a bit set for each cluster DCHECK finds as “previously allocated” or “in file structure but not in allocation map.” DCHECK then makes another pass through the directory structure to determine the pathlists for these questionable clusters. You can save the bitmap work files by specifying the *-m* option on the command line.
- For best results, DCHECK should have exclusive access to the disk being checked. Otherwise, the command might be fooled by a change in the disk allocation map while DCHECK is building a bitmap file. DCHECK cannot process disks with more than 39 levels of directories.
- *-p* causes DCHECK to make a second pass through the file structure and print pathlists for clusters that are not in the allocation map but are allocated or existing in a file structure.

-w tells DCHECK where to place its allocation map work file(s). The specified pathlist must be a full pathlist for a directory. (DCHECK uses directory /D0 if you do not specify *-w*.) If you doubt the structure integrity of the diskette being checked, do not place the allocation map work files on that diskette.

Examples:

- The following two examples demonstrate DCHECK sessions:

```
dcheck /d2 
```

A sample screen display might be:

```
Volume - 'My system disk' on device /d2
$009A bytes in allocation map
1 sector per cluster
$000276 total sectors on media
Sector $000002 is start of Root directory
FD
$0010 sectors used for id, allocation map
and Root directory
Building allocation map work file...
Checking allocation map file...

'My system disk' file structure is intact
1 directory
2 files
```

```
dcheck -mpw=/d2 /d0 
```

A sample screen display might be:

```
Volume - 'System diskette' on device /d0
$0046 bytes in allocation map
1 sector per cluster
$00022A total sectors on media
Sector $000002 is start of Root directory
FD
$0010 sectors used for id, allocation map
and Root directory
Building allocation map work file...
Cluster #00040 was previously allocated
*** Bad FD segment ($111111-$23A6F0) for
file: /D0/TEXT/junky.file
Checking allocation map file...
Cluster $000038 in file structure but not
in allocation map
Cluster $00003B in file structure but not
in allocation map
Cluster $0001B9 in allocation map but not
in file structure
Cluster $0001BB in allocation map but not
in file structure
```

Pathlists for questionable clusters:
Cluster \$000038 in path: /d0/DS9boot
Cluster \$00003B in path: /d0/DS9boot
Cluster \$000040 in path: /d0/DS9boot
Cluster \$000040 in path: /d0/test/
double.file

1 previously allocated clusters found
2 clusters in file structure but not in
allocation map
2 clusters in allocation map but not in
file structure
1 bad file descriptor sector

'System diskette' file structure is not
intact
5 directories
25 files

DEINIZ

Syntax: `deiniz devname [...]`

Function: Deinitializes and detaches a device.

Parameter:

devname The name of one or more devices you want to deinitialize.

Notes:

- Use DEINIZ with INIZ. For example, you can use INIZ to initialize a window, then redirect information to the window. View the information by pressing until it appears. When you no longer need the window, use DEINIZ to remove the window and return its memory to the system.
- DEINIZ performs an OS-9 I\$Detach call for all specified devices.

Example:

To deinitialize the /w1 (Window 1) device after it has been initialized, type:

```
deiniz w1 
```

DEL

Syntax: `del [-x] filename [...]`

Function: Deletes the file(s) specified.

Parameter:

filename The name of the file to delete. Include as many filenames as you want.

Option:

`-x` Causes DEL to assume the file is in the current execution directory.

Notes:

- You can delete only files for which you have write permission.

You can delete a directory in two ways: (1) Delete all the files in the directory, change it to a non-directory file using ATTR, then use DEL to remove the directory, or (2) Use the DELDIR command.

- The following example shows what appears on the screen when you display a directory, delete one of the directory's files, then display the directory again:

```
dir /d1 

directory of /d1 14.29.46
myfile      newfile

del newfile 
dir /d1 

directory of /d1 14.30.37
myfile
```

Examples:

- To delete files named Text_program and Test_program, type:

```
del text_program test_program 
```

- To delete a file on a drive other than the current working drive, use a complete pathlist, such as:

```
del /d1/number_five 
```

- To delete a file named Cmds.subdir in the current execution directory, type:

```
del -x cmds.subdir 
```

DELDIR

Syntax: `deldir dirname`

Function: Deletes all subdirectories and files in a directory; then, deletes the directory itself.

Parameter:

dirname The pathlist to the directory you want to delete.

Notes:

- DELDIR is a convenient alternative to individually deleting all the files and subdirectories from a directory before deleting the directory itself.
- When DELDIR runs, it displays a prompt after the command line:

```
deldir oldfiles 
Deleting directory file.
List directory, delete directory, or quit ?
(l/d/q)
```

Pressing causes a DIR E command to run so you can see the directory files before DELDIR removes them.

Pressing starts the deletion process.

Pressing cancels the command.

- The directory to be deleted might include other directories, which in turn might include other directories, and so forth. In this case, DELDIR begins with the lower directories and works its way upward.

You must have write permission to delete any files and directories in this substructure. If not, DELDIR terminates when it encounters the first file for which you don't have write permission.

- DELDIR automatically calls DIR and ATTR. Therefore, these files must reside in the current execution directory.

DIR

Syntax: `dir [opts][dirname or pathlist]`

Function: Displays a formatted list of filenames in a directory. The output format adjusts itself for 80- or 32-column displays.

Parameters:

<i>dirname</i>	The name of the directory you want to view.
<i>pathlist</i>	The pathlist to the directory you want to view.
<i>opts</i>	Either or both of the following options.

Options:

If you don't specify any parameters, DIR shows the current data directory.

x	Displays the current execution directory.
e	Displays the entire description for each file: size, address, owner, permissions, date and time of last modification.

Examples:

- To display the current data directory, type:

```
dir 
```

- To display the current execution directory, type:

```
dir x 
```

- To display the entire description of all files in the current execution directory, type:

```
dir x e 
```

- To display the parent of the current data directory, type:

```
dir .. 
```

- To display a directory named NEWSTUFF, type:

```
dir newstuff 
```

- Following is a sample 80-column DIR display using the e option:

```
dir e 
```

The screen might display:

Directory of . 16:50:12

Owner	Last modified	Attributes	Sector	Bytecount	Name
2F	85/05/20 1631	-----wr	A	3AGC	OS9Boot
0	85/05/20 1345	d-ewrewr	48	640	CMDS
0	85/05/20 1350	d-ewrewr	177	A0	SYS
0	85/05/20 1351	----r-wr	192	E	startup
0	85/05/20 1351	d-ewrewr	194	E0	DEFS

- Following is an 80-column DIR display using no options:

```
dir 
```

The screen might display:

Directory of . 16:50:37

```
OS9Boot    CMDS    SYS    startup
DEFS
```

- Following is a 32-column DIR display using the e option:

```
dir e 
Directory of . 16:52:04

Modified on   Owner      Name
   Attr      Sector    Size
=====
85/05/20  1643      2F    OS9Boot
-----wr           A      3A6C
85/05/20  1345      0     CMDS
d-ewrewr           48     640
85/05/20  1350      0     SYS
d-ewrewr           177    A0
85/05/20  1351      0    startup
----r-wr           192    E
85/05/20  1351      0     DEFS
d-ewrewr           194    E0
```

- Following is a 32-column DIR display using no options:

```
dir 

Directory of . 16:52:29
OS9Boot  CMDS    SYS
startup  DEFS
```

DISPLAY

Syntax: `display hex [...]`

Function: Reads one or more hexadecimal numbers (you type as parameters), converts them to ASCII characters, and writes them to the standard output (normally the screen).

Parameters:

hex A list of one or more hexadecimal numbers.

Notes:

- Use DISPLAY to send special characters (such as cursor and screen control codes) to terminals and other I/O devices.
- Following is an example of a command and the resulting output. ABCDEF are ASCII characters corresponding to hex 41 42 43 44 45 46.

```
display 41 42 43 44 45 46   
ABCDEF
```

Examples:

- To reroute a *form feed* (hex 0C) to the printer, type:

```
display 0C >/p 
```

- To ring the *bell* through the video speaker, type:

```
display 07 
```

DSAVE

Syntax: `dsave [opts][devname][dirname] > pathlist`

Function: Copies or *backs up* all files in one or more directories.

Parameters:

<i>devname</i>	The drive on which the source directory exists. If you do not specify <i>devname</i> DSAVE assumes Drive /D0.
<i>dirname</i>	The name of the destination directory. Use CHD to make the current directory the directory to receive the copies.
<i>pathlist</i>	A command procedure file in which DSAVE stores its output.
<i>opts</i>	One or more of the following options.

Options:

-b	Makes the destination or target diskette a system diskette by copying the source diskette's OS9Boot file, if present.
-i	Indents for directory levels.
-l	Tells DSAVE not to process directories below the current level.
-m	Tells DSAVE not to include MAKDIR commands in the procedure file it creates.
-sinteger	Sets memory for the copy parameter to <i>integer</i> kilobytes.
-v	Verifies copies by forking to CMP after copying each file.

Notes:

- DSAVE does not directly affect the system. Instead, it generates a procedure file that you execute later to do the work.
- When you run DSAVE, it creates a procedure file (a file of commands). You then execute the newly created file by typing its pathlist. The procedure contains all the commands to create and change directories as needed in order to copy the specified directory. DSAVE copies the files in the current data directory. It also copies the current data directory subdirectories, unless you specify the -l option.
- To use DSAVE, first change the data directory to the directory you wish to copy. Execute DSAVE by specifying the drive from which to copy and then redirecting output to a file to receive the copy commands. Be sure to name a file that does not already exist.

When DSAVE completes the procedure, use CHD to change to the data directory to receive the copied files. Then, execute the procedure file.

- If DSAVE encounters a directory file, it automatically includes MAKDIR and CHD commands in the output before generating COPY commands for files in the subdirectory. The procedure file exactly replicates all levels of the file system from the current data directory downward.
- If the current data directory is the ROOT directory of the disk, DSAVE creates a procedure file that backs up the entire disk file by file. This is useful when you need to copy a number of files from either disks formatted differently or from floppy diskettes to a hard disk.

Examples:

- In the following series of commands, CHD positions you in the ROOT directory of /D2, the directory to be copied. Then, DSAVE makes the procedure file Makecopy. Using CHD /D1 causes the copy to go in the /D1 ROOT directory. The final command executes the procedure file.

```
chd /d2   
dsave /d2 >/d0/makecopy   
chd /d1   
/d0/makecopy 
```

- The following command copies all files from /D0 to /D1. It pipes the procedure file output of DSAVE into a shell for immediate execution.

```
dsave /d0 /d1 ! shell 
```

- The following command lets you view the output generated by a DSAVE command. It uses 48 kilobytes of memory and indents directories. Because output goes to the screen, this command does not create a procedure file to copy any files:

```
dsave -s48 -i 
```

- This command operates in the same manner as the previous command. However, because it specifies a procedure file pathlist, it stores the generated commands in a procedure file rather than displaying them on the screen:

```
dsave -s48 -i > copyfile 
```

ECHO

Syntax: `echo text`

Function: Echoes text to the screen.

Parameters:

text The character or characters you type.

Notes:

- Use ECHO to generate messages in shell procedure files or to send an initialization character sequence to a terminal. The text should not include punctuation characters used by the shell.
- The following example prints the message LISTING ERROR MESSAGES to the screen and lists the file SYS/errmsg to the printer as a background task.

```
echo LISTING ERROR MESSAGES; list sys/  
errmsg >/p& 
```

Examples:

- To display a message on the screen, type:

```
echo This text is echoing 
```

- To echo text to the console, type:

```
echo >/term **WARNING DATA ON DISK WILL BE  
LOST 
```

- The following combines the ECHO and LIST commands to echo the entered text to the printer and to direct the contents of the Trans file to the printer.

```
echo >/p LISTING OF TRANSACTION; list trans  
>/p& 
```

ERROR

Syntax: `error errnumber [...]`

Function: Displays the text error message that corresponds with the specified OS-9 error number.

Parameters:

errnumber Is an OS-9 error code in the range 1-255.

Notes:

- ERROR opens the Errmsg file in the SYS directory and reads through the file for an error code that matches the specified number. It then displays the text that corresponds to the error code.
- The Errmsg file contains descriptions of the standard OS-9 errors. The order of the file is arranged to provide quick access to operation system error descriptions.

Example:

- To display a description of the OS-9 error Numbers 215 and 216, type:

```
error 215 216 
```

The screen displays:

```
215 - Bad Path Name
216 - Path Name Not Found
```

EX

Syntax: `ex filename`

Function: Starts a process by *chaining* from the current shell to the new process. Chaining means that execution control is turned over to the new process.

Parameters:

filename The name of the program or module you want to execute.

Notes:

- Because EX is a built in Shell command, it does not appear in the CMDS directory.
- Using EX causes the shell from which you are operating to terminate. If the new process also terminates and you do not have another shell running on another terminal or window, OS-9 is left without any processes, and you must reboot your computer and OS-9.
- If a shell is running on another window or device, you can restart a new shell from that window or device. For instance, if you use EX to initialize BASIC09 from /TERM then exit BASIC09, /TERM is dead and cannot accept keyboard input. However, if you also have a shell operating in a window, you can type the following from that window:

```
shell i=/term& 
```

This reinitializes a shell on /TERM. It can now accept keyboard input and OS-9 commands.

- Use EX to save memory when the shell is not needed, for instance when using BASIC09.

- If you use EX on a command line with other commands, it must be the last command. Any commands following EX are not processed.

Example:

- To run BASIC09 without a resident shell, type:

```
ex basic09 
```

FORMAT

Syntax: `format devname [name] [opts]`

Function: Establishes and verifies an initial file structure on a floppy diskette or a hard disk. You must format all disks before you can use them on an OS-9 system.

Parameters:

<i>devname</i>	The drive name of the disk you want to format.
<i>name</i>	The name you want to assign the newly formatted disk. Enclose the disk name in double quotation marks.
<i>opts</i>	One or more of the following options.

Options:

l	Writes system format information only, does not physically format disk.
r	Causes the format to proceed automatically, without issuing prompts.
1	Formats single-sided. Use with single-sided drives or single-sided diskettes in double-sided drives.
2	Causes a double-sided format. Use with double-sided drives and double-sided diskettes.
' <i>cylinders</i> '	The number of cylinders (in decimal) that you want formatted.
: <i>interleave</i> :	The number of the sector interleave value (in decimal).

Notes:

- Be sure the disk you want to format is NOT write-protected. Otherwise, FORMAT generates error code #242 (write protect), and the system returns to the OS-9 prompt without formatting the diskette.

- If you are formatting a hard disk, first type:

```
tmode -pause 
```

This command turns off the screen pause function. Otherwise, the process stops whenever the sector verification process fills the display screen. If you forget to turn off the screen pause, press the space bar whenever the screen fills. Execution then continues.

When formatting finishes, type:

```
tmode pause 
```

This re-establishes the screen pause function.

- The formatting process works this way:
 1. FORMAT physically initializes a disk and divides its surface into sectors.
 2. FORMAT reads back and verifies each sector. If a sector fails to verify after several attempts, FORMAT excludes it from the initial free space on the diskette. As verification proceeds, the process displays track numbers.
 3. FORMAT writes the disk allocation map, ROOT directory, and identification sector to the first few sectors of Track 0. These sectors must not be defective.
- FORMAT asks for a disk volume name, which can be up to 32 characters long and can include spaces or punctuation. (Later, you can use the FREE command to display the name.)

- For step-by-step instructions on formatting, refer to *Getting Started With OS-9*.

Examples:

- To format a diskette in Drive /D1, type:

```
format /d1 
```

- To format a diskette in Drive /D1 with the name Test Disk and without prompts, type:

```
format /d1 r "test disk" 
```

- To format hard Disk /H0, type:

```
tmode -pause   
format /h0   
tmode pause 
```

- To format a double-sided diskette in Drive /D2 with 27 cylinders and the name Database, type:

```
format /d1 2 "database" '27' 
```

FREE

Syntax: `free [devname]`

Function: Displays the number of unused sectors (256-byte storage areas) on a disk drive. These sectors are available for new files or for expanding existing files.

Parameters:

devname The disk drive for which you want to display the number of free sectors.

Notes:

- The device name you specify must be a disk drive. FREE also displays the disk's name, creation date, and cluster size. If you don't specify a drive, FREE selects the drive that contains the current data directory.
- The cluster size for the Color Computer is one sector.

Examples:

- To display the number of free sectors on the current disk, type:

```
free 
```

The screen is similar to this:

```
"COLOR COMPUTER DISK" created on: 83/05/28
Capacity: 630 sectors (1-sector clusters)
15 Free sectors, largest block 12 sectors
```

- To display the number of free sectors on the diskette in Drive /D1, type:

```
free /d1 
```

A sample screen display is:

```
"DATA DISK" created on: 83/06/16  
Capacity: 630 sectors (1-sector clusters)  
445 Free sectors, largest block 442 sectors
```

HELP

Syntax: help [*command name*] [-?]

Function: Displays the use and syntaxes of OS-9 commands.

Parameters:

<i>command</i>	The command(s) for which you want help.
<i>name</i>	Include as many command names as you want.
-?	Gives a list of help topics.

Notes:

- HELP uses a file named Helpmsg, which is located in the SYS directory on your system diskette.

Examples:

- To obtain help for the BACKUP command, type:

```
help backup 
```

The screen displays:

```
Syntax: backup [e][s][-v][dev][dev]
Usage: Copies all data from one device to
another
```

- If you try to obtain help for a non-existent command, HELP displays an error message. For instance, if you type:

```
help me 
```

```
me: no help available
```

- You can also obtain help for the HELP command. To do so, type:

```
help help 
```

The screen displays:

```
Syntax: Help [subject][-?]
Usage: Give on-line help to users
       Will prompt if no subjects given
Opts: -? give list of help topics
```

IDENT

Syntax: `ident filename [opts]`

Function: Displays header information for memory modules.

Parameters:

- | | |
|-----------------|---|
| <i>filename</i> | The name of the file or module for which you want to view identification information. |
| <i>opts</i> | One or more of the following options. |

Options:

- | | |
|----|--|
| -m | Causes IDENT to assume that <i>filename</i> is a module in memory |
| -v | Tells IDENT not to verify module cyclic redundancy check |
| -x | Causes IDENT to assume that <i>filename</i> is in the execution directory |
| -s | Displays the following module information on a single line: the edition byte (first byte after module name), the type/language byte, the module CRC and the module name. A period (.) indicates that the CRC verifies. A question mark (?) indicates that the CRC does not verify. |

Notes:

- IDENT displays the module size, CRC bytes (with verification), and—for program and device driver modules—the execution offset and the permanent storage requirement bytes.

- IDENT displays and interprets the type/language and attribute revision bytes. IDENT displays the byte immediately following the module name because most Microware®-supplied modules set this byte to indicate the module edition.
- IDENT displays all modules contained in a disk file.

Examples:

- To display header information for a file named Ident that resides in computer memory, type:

```
ident -m ident 
```

The screen might display:

```
Header for: IDENT
Module size: $06E7          #1767
Module CRC: $540BB2 (Good)
Hdr parity: $C9
Exec. off: $0230           #573
Data Size: $099C          #2460
Edition:                   $07          #7
Ty/La At/Rv: $11 $81
Prog mod, 6809 obj, re-en, R/O
```

In the example, Hdr parity = header parity; Exec. off = execution offset; Data size = permanent storage requirements; Edition = first byte after module name; Ty/La At/Rv = type/language attribute/revision; and Prog mod, 6809 obj, re-en = module type, language, attribute.

- To display header information for the OS9Boot file, type:

```
ident /D0/OS9boot -s 
```

The display might include:

```
17 $C0 $F2922F . OS9p2
67 $C0 $0B2322 . Init
12 $C1 $2E9EDB . IDMan
27 $D1 $B665E3 . RBF
 6 $E1 $055580 . CC3Disk
82 $F1 $FC1918 . D0
82 $F1 $9F4210 . D1
82 $F1 $E6B118 . DD
11 $D1 $10A3FA . SCF
14 $E1 $8524F1 . CC3IO
 1 $C1 $B53D94 . VDGInt
 3 $C1 $792B7E . GrfInt
83 $F1 $AB5AE5 . TERM
83 $F1 $7AB2DB . W
83 $F1 $C3E38A . W1
83 $F1 $948878 . W2
83 $F1 $36016B . W3
83 $F1 $0AE2B6 . W4
83 $F1 $123B9A . W5
83 $F1 $1CF164 . W6
83 $F1 $B71DF5 . W7
11 $E1 $C8F073 . ACIAPAK
82 $F1 $9E655D . T2
12 $E1 $CC3EA4 . PRINTER
83 $F1 $FE3BAE . P
 4 $D1 $AD6718 . PipeMan
 2 $E1 $5B2B56 . Piper
80 $F1 $CC06AF . Pipe
 9 $C1 $BE93F4 . Clock
 3 $11 $CA1F99 . CC3Go
```

Since the `-s` option appears in the command line, `IDENT` displays each module's information on a single line. In the first line of the output, for instance, 17 = edition byte (first byte after name), `$C0` = type/language byte, `$A366DC` = CRC value, `.` = OK CRC check, and `OS9p2` = module name.

INIZ

Syntax: `iniz devname [...]`

Function: Initializes the specified device driver.

Parameters:

devname The name of one or more devices to initialize.

Notes:

- You can use INIZ in the Startup file or at the system start-up to initialize devices and allocate their static storage at the top of memory to reduce memory fragmentation.
- INIZ attaches the specified device to OS-9, places the device address in a new device table entry, allocates the memory needed by the device driver, and calls the device driver initialization routine. INIZ does not reinitialize a device that you or the system previously installed.
- If you change the printer (/p) to a non-shareable device (a device that is not re-entrant), do not initialize it with INIZ.

Examples:

- To initialize the /P (printer) and /T2 (terminal 2) devices, type:

```
iniz p t2 
```

KILL

Syntax: `kill procID`

Function: Terminates the process specified by *procID*.

Parameters:

procID The ID number of the process to kill.

Notes:

- Unless you are the *Super User* (User Number 0), you can only terminate a process that has your user number. (Use PROCS to obtain the process ID numbers.) The Super User can terminate any process.
- If a process is waiting for I/O, you cannot cancel it until the current I/O operation terminates. Therefore, if you KILL a process and PROCS shows it still exists, it is probably waiting to receive a line of data from a terminal.
- Because KILL is a built-in shell command, it does not appear in the CMDS directory.

Examples:

- To KILL the process with the ID number 5, type:

```
kill 5 ENTER
```

- The following commands: (1) use PROCS to determine that the ID number of the process to be killed is 3, (2) terminate process 3, and (3) use PROCS to confirm that KILL has cancelled the process.

procs

		User				Mem Stack			
Id	PId	Number	Pty	Age	Sts	Signl	Siz	Ptr	Primary Module

2	1	0	128	128	980	0	3	\$78B2	Shell
3	5	0	128	128	980	0	2	\$74AC	Tsmon
4	5	0	128	128	980	0	6	\$05F3	Procs
5	0	0	128	129	980	0	3	\$6FE2	Shell

kill 3

procs

		User				Mem Stack			
Id	PId	Number	Pty	Age	Sts	Signl	Siz	Ptr	Primary Module

2	1	0	128	128	980	0	3	\$78B2	Shell
3	5	0	128	128	980	0	6	\$05F3	Procs
5	0	0	128	129	980	0	3	\$6FE2	Shell

LINK

Syntax: `link modname`

Function: Locks a previously loaded module into memory.

Parameters:

modname The name of the memory module to link.

Notes:

- If the module is not already in memory, you must use **LOAD** prior to using **LINK**. The link count of the module increases by one each time the system *links* it. Use **UNLINK** to *unlock* the module when you no longer need it.

Examples:

- To lock the Edit module into memory, type:

```
link edit 
```

LIST

Syntax: `list filename [...]`

Function: Lists the contents of a text file or files.

Parameters:

filename The name of the file you want to list. Include as many filenames on one line as you want, up to the maximum line length of 199 characters.

Notes:

- LIST copies text lines from a file to the standard output path. The program terminates upon reaching the end-of-file of the last input path. If you specify more than one file, LIST copies the files in the order in which you list them.
- Use LIST to examine or print text files.
- Do not LIST executable files. Doing so can cause your system to lock or crash. To view executable files, use DUMP.

Examples:

- To list the contents of the Startup file to the printer, type:

```
list /d0/startup >/p& 
```

The ampersand makes the printing job a concurrently executed task.

- To list three files to the screen, type:

```
list /d1/user5/document /d0/myfile /d0/  
bob/text 
```

- To copy everything you type at the keyboard to the printer, type:

```
list /term >/p 
```

To go back to the standard output path (the video display) press **CTRL** **BREAK**.

- The following commands create a file called `Animals`, consisting of six entries. `LIST`, with the filename `Animals` as a parameter, displays the contents of the new file.

```
build animals ENTER
? cat ENTER
? cow ENTER
? dog ENTER
? elephant ENTER
? bird ENTER
? fish ENTER
? ENTER

list animals ENTER
```

The screen displays:

```
cat
cow
dog
elephant
bird
fish
```

LOAD

Syntax: `load pathlist`

Function: Loads a module (program) from a file into memory.

Parameters:

pathlist Specifies the module to load.

Notes:

- LOAD opens the path you specify, then loads into memory one or more modules from it. The process adds the names of the new modules to the module directory. If LOAD finds that a specified module has the same name and type as a module already in memory, it keeps the module with the highest revision level.
- If the pathlist for LOAD does not include a drive name, LOAD uses the current execution directory. To LOAD a module from a directory other than the current execution directory, specify a full pathlist, beginning with a drive name if applicable.

Examples:

- In the following example, MDIR displays the names of modules currently resident in memory. Then, LOAD loads the Edit module into memory. MDIR again lists memory modules, and this time shows that Edit is successfully added to memory.

```
mdir 
```

The screen display is similar to the following:

```
Module Directory at 12:49:52
REL      Boot      OS9p1      OS9p2      Init
IOMan    RBF      CC3Disk    D0          D1
DD       SCF      CC310     VDGInt     GrfInt
TERM     W         W1        W2          W3
W4       W5        W6        W7          ACIAPAK
T2       PRINTER  P         PipeMan    Piper
Pipe     Clock    CC3Go     CC3HDisk   H0
Shell    Copy     Date      DeIniz     Del
Dir      Display  Echo      Iniz       Link
List     Load    MDir     Merge      Mfree
Procs    Rename   Setime    Tmode      Unlink
Basic09  GrfDrv
```

```
load edit 
```

```
mdir 
```

The screen displays:

```
Module Directory at 12:51:12
REL      Boot      OS9p1      OS9p2      Init
IOMan    RBF      CC3Disk    D0          D1
DD       SCF      CC310     VDGInt     GrfInt
TERM     W         W1        W2          W3
W4       W5        W6        W7          ACIAPAK
T2       PRINTER  P         PipeMan    Piper
Pipe     Clock    CC3Go     CC3HDisk   H0
Shell    Copy     Date      DeIniz     Del
Dir      Display  Echo      Iniz       Link
List     Load    MDir     Merge      Mfree
Procs    Rename   Setime    Tmode      Unlink
Basic09  GrfDrv   Edit
```

MAKDIR

Syntax: `mkdir pathlist or dirname`

Function: Creates a directory according to the pathlist given. You must have write permission for the parent directory of the directory you are creating.

Parameters:

- | | |
|-----------------|---|
| <i>pathlist</i> | The path to the directory you want to create. |
| <i>dirname</i> | The name of the directory you want to create. |

Notes:

- When MAKDIR initializes the new directory, the directory contains only the “.” and “..” files.
- MAKDIR enables all permissions for the directory it creates.
- To follow OS-9 convention, capitalize all directory names.

Examples:

- To create a directory on Drive /D1, use the directory’s full pathlist from the root, such as:

```
mkdir /d1/STEVE/PROJECT 
```

- To create a directory called DATAFILES within the current data directory, type:

```
mkdir DATAFILES 
```

- To create a directory called SAVEFILES in the parent of the current directory, type:

```
mkdir ../SAVEFILES 
```

MDIR

Syntax: `mdir [e]`

Function: Displays the names of modules currently in memory. MDIR automatically adjusts its output for 32- or 80-column displays.

Options:

- e** Causes a full listing of the extended physical address (block number and offset within the block), size, type, revision level, re-entrant attribute, user count, and name of each module. MDIR shows numbers in hexadecimal. The display adjusts for 80 or 32 columns.

Notes:

- Many of the modules displayed by MDIR are OS-9 system modules and are not executable as programs. **Always check the module type code before running a module with which you are not familiar.**

Examples:

- Because MDIR adjusts to either 32 or 80 columns, the following command produces a full module listing in either format:

```
mdir e 
```

The 80-column display of MDIR e is:

```
Module Directory at 03:03:53

Block Offset Size Typ Rev Attr Use Module Name
-----
3F D06 12A C1 1 r... 0 REL
```

System Command Descriptions / 6

3F	E30	1D0	C1	1	r...	1	Boot
3F	1000	ED9	C0	8	r...	0	OS9p1
1	200	CA1	C0	2	r...	1	OS9p2
1	EA1	2E	C0	1	r...	1	Init
1	ECF	993	C1	1	r...	1	IDMan
1	1862	122B	D1	1	r...	6B	RBF
1	2A8D	476	E1	1	r...	2	CC3Disk
1	2F03	30	F1	1	r...	2	D0
1	2F33	30	F1	1	r...	0	D1
1	2F63	30	F1	1	r...	0	DD
1	2F93	5B6	D1	1	r...	22	SCF
1	3549	B91	E1	1	r...	D	CC3ID
1	40DA	CE7	C1	1	r...	1	VDGInt
1	4DC1	BF2	C1	1	r...	1	GrfInt
1	59B3	45	F1	1	r...	8	TERM
1	59F8	42	F1	1	r...	0	W
1	5A3A	43	F1	1	r...	0	W1
1	5A7D	43	F1	1	r...	0	W2
1	5AC0	43	F1	1	r...	0	W3
1	5B03	43	F1	1	r...	0	W4
1	5B46	43	F1	1	r...	0	W5
1	5B89	43	F1	1	r...	0	W6
1	5BCC	43	F1	1	r...	5	W7
1	5C0F	3B5	E1	1	r...	8	ACIAPAK
1	5FC4	3F	F1	1	r...	9	T2
1	6003	17A	E1	1	r...	D	PRINTER
1	617D	3C	F1	1	r...	D	P
1	61B9	219	D1	1	r...	12	PipeMan
1	63D2	28	E1	1	r...	12	Piper
1	63FA	26	F1	1	r...	12	Pipe
1	6420	174	C1	1	r...	1	Clock
1	6594	1AA	11	1	1	CC3Go
6	0	5F2	11	1	r...	3	Shell
6	5F2	2DC	11	1	r...	0	Copy
6	8CE	FD	11	1	r...	0	Date
6	9CB	76	11	1	r...	0	DeIniz
6	A41	A5	11	1	r...	0	Del
6	AE6	365	11	1	r...	0	Dir
6	E4B	84	11	1	r...	0	Display
6	ECF	22	11	1	r...	0	Echo
6	EF1	6A	11	1	r...	0	Iniz
6	F5B	2C	11	1	r...	0	Link
6	F87	4F	11	1	r...	0	List
6	FD6	24	11	1	r...	0	Load
6	FFA	2F1	11	1	r...	1	MDir

OS-9 Commands Reference

6	12EB	68	11	1 r...	0 Merge
6	1353	1EB	11	1 r...	0 Mfree
6	153E	319	11	1 r...	0 Procs
6	1857	11D	11	1 r...	0 Rename
6	1974	118	11	1 r...	0 Setime
6	1A8C	301	11	1 r...	0 Tmode
6	1D8D	2D	11	1 r...	0 Uniink

- The 32-column display of MDIR is:

Module Directory at 03:06:49

Blk	Ofst	Size	Ty	Rv	At	Uc	Name
3F	D06	12A	C1	1 r		0	REL
3F	E30	1D0	C1	1 r		1	Boot
3F	1000	ED9	C0	8 r		0	OS9p1
1	200	CA1	C0	2 r		1	OS9p2
1	EA1	2E	C0	1 r		1	Init
1	ECF	993	C1	1 r		1	IDMan
1	1862	122B	D1	1 r		70	RBF
1	2A8D	476	E1	1 r		2	CC3Disk
1	2F03	30	F1	1 r		2	D0
1	2F33	30	F1	1 r		0	D1
1	2F63	30	F1	1 r		0	DD
1	2F93	5B6	D1	1 r		24	SCF
1	3549	B91	E1	1 r		D	CC310
1	40DA	CE7	C1	1 r		1	VDGInt
1	4DC1	BF2	C1	1 r		1	GrfInt
1	59B3	45	F1	1 r		8	TERM
1	59F8	42	F1	1 r		0	W
1	5A3A	43	F1	1 r		0	W1
1	5A7D	43	F1	1 r		0	W2
1	5AC0	43	F1	1 r		0	W3
1	5B03	43	F1	1 r		0	W4
1	5B46	43	F1	1 r		0	W5
1	5B89	43	F1	1 r		0	W6
1	5BCC	43	F1	1 r		5	W7
1	5C0F	3B5	E1	1 r		A	ACIAPAK
1	5FC4	3F	F1	1 r		B	T2
1	6003	17A	E1	1 r		D	PRINTER
1	617D	3C	F1	1 r		D	P
1	61B9	219	D1	1 r		12	PipeMan
1	63D2	28	E1	1 r		12	Piper
1	63FA	26	F1	1 r		12	Pipe
1	6420	174	C1	1 r		1	Clock

1	6594	1AA	11	1	.	1	CC3Go
6	0	5F2	11	1	r	3	Shell
6	5F2	2DC	11	1	r	0	Copy
6	8CE	FD	11	1	r	0	Date
6	9CB	76	11	1	r	0	DEIniz
6	A41	A5	11	1	r	0	Del
6	AE6	365	11	1	r	0	Dir
6	E4B	84	11	1	r	0	Display
6	ECF	22	11	1	r	0	Echo
6	EF1	6A	11	1	r	0	Iniz
6	F5B	2C	11	1	r	0	Link
6	F87	4F	11	1	r	0	List
6	FD6	24	11	1	r	0	Load
6	FFA	2F1	11	1	r	1	MDir
6	12EB	68	11	1	r	0	Merge
6	1353	1EB	11	1	r	0	Mfree
6	153E	319	11	1	r	0	Procs
6	1857	11D	11	1	r	0	Rename
6	1974	118	11	1	r	0	Setime
6	1A8C	301	11	1	r	0	Tmode
6	1D8D	2D	11	1	R	0	Unlink

MERGE

Syntax: `merge [filename][...]`

Function: Copies files to the standard output path. By redirecting the output of the MERGE command, you can combine several files into one file, or direct several files to the printer.

Parameters:

filename Specifies the files to combine.

Notes:

- Use MERGE to combine several files into a single output file. It copies data in the order in which you type the filenames.
- MERGE does not output line editing characters (such as the automatic line feed).
- You normally use MERGE with the standard output redirected to a file or device.
- You can use MERGE to append or copy any type or mixture of files to another device.

Examples:

- To merge four files into a new file called Combined.file, and send the results to the new file instead of to the video display, type:

```
merge file1 file2 file3 file4 >combined.file  

```

- To merge two files, and send the output to the printer, type:

```
merge compile.list asm.list >/P 
```

MFREE

Syntax: mfree

Function: Displays a list of memory areas not presently in use and, therefore, available for assignment.

Notes:

- MFREE displays the block number, physical (extended) beginning and ending addresses, and the size of each contiguous area of unassigned RAM. It gives the size in number of blocks and in kilobytes. The block size is 8 kilobytes per block. Free memory for user data areas does not need to be contiguous because the MMU can map scattered free blocks to be logically contiguous.

Examples:

- Type this command:

```
mfree 
```

The screen shows a display similar to this:

```
Blk Begin  End  Blks  Size
-----
10 12000 10FFF    1    8K
18 18000 1DFFF    3   24K
20 20000 3FFFF   16  128K
      Total:    20  160
```

MODPATCH

Syntax: `modpatch [options] filename [options]`

Function: modifies modules residing in memory. MODPATCH reads a *patchfile* and executes the commands in the patchfile to change the contents of one or more modules.

Parameters:

<i>filename</i>	The name of a file containing instructions for MODPATCH
<i>options</i>	One of the following options that change MODPATCH's function

Options:

-s	Silent mode, does not display patchfile command lines as they are executed.
-w	Does not display warnings, if any
-c	Compares only, does not change the module

Notes:

- Before using MODPATCH, you must create a patchfile to supply the data to control MODPATCH's operation. This file contains single-letter commands and the appropriate module addresses. The commands are:

l <i>modulename</i>	Link to the module specified by <i>modulename</i> .
c <i>offset original newval</i>	Change the byte at the offset address specified by <i>offset</i> from the value specified by <i>original</i> to the new value specified by <i>newval</i> . If the original value does not match <i>original</i> , MODPATCH displays a message.
v	Verify the module—update the modules CRC. If you plan to save the patched module to a file that the system can load, you must use this command.
m	Mask IRQ's. Turns off interrupt requests (for patching service routines).
u	Unmask IRQ's. Turns on interrupt requests (for patching service routines).

- You can use the BUILD command or any word processing program to create patchfiles.
- Module byte addresses begin at 0. MODPATCH changes values pointed to by an offset address (offset from 0) rather than an absolute memory address.

- To view the contents of a memory module, use `SAVE` and `DUMP` to copy the module to a file and display its contents. Also use `SAVE` to copy the patched module to a disk file.
- Changing a memory module might not produce an immediate effect. You have to duplicate the initialization procedure for that module. This means, if the module loads during bootup, you have to create a new boot file that includes the changed module, then reboot using the new boot file.
- To use the patched module in future system boots, use `SAVE` to store the module in the `MODULES` directory of your system disk. You can then use `OS9GEN` to create a new system disk using the patched module. If you are using the patched module to replace another module, rename the original module and then give the patched module the original name.
- If you patch a module that is loaded during the system boot, you can use `COBBLER` to make a new system boot that uses the patched module.

Examples:

The following example shows the commands, the screen prompts, and the entries you make to patch the standard 40-column term window descriptor to be an 80-column screen rather than the standard 40-column screen:

```
OS9:build termpatch 
? I term 
? c 002c 28 50 
? c 0030 01 02
? v 
? 
OS9: modpatch termpatch 
```

To change the size, columns, and colors of Device Window W1, create the following procedure file and name it W180:

```
I w1
c 0030 01 02
c 002c 1b 50
c 002d 0b 18
```

If the W1 module is not already in memory, load it from the MODULES directory of your system disk. Then, before initializing W1, run MODPATCH:

```
modpatch w180 
```

Next, initialize W1:

```
iniz w1 
shell i=/w1& 
```

Press to display the new window with 80 columns, 24 lines, and a white background.

MONTYPE

Syntax: `montype type`

Function: Sets your system for the type of monitor you are using

Parameters:

Parameters:

<i>type</i>	A single letter indicating the monitor type:
<code>c</code>	for composite monitors or color televisions
<code>r</code>	for RGB monitors
<code>m</code>	for monochrome monitors or black and white televisions

Notes:

- Different types of color monitors display colors differently. For the best results, set your system to the type of monitor you are using.
- If you are using a monochrome monitor or black and white television, you can obtain a sharper image by setting your monitor type to monochrome.
- Include the MONTYPE command in your system's Startup file to automatically boot in the proper monitor mode.
- If you do not use MONTYPE, the system defaults to `c` (composite monitor).

Example:

To set your system for an RGB monitor, type:

```
montype r 
```

To add a MONTYPE command to your existing Startup file, first use BUILD to create the new command. For example:

```
build temp   
montype r   

```

Next, append the file to Startup. Type:

```
merge startup temp > startup.new 
```

Delete the temp file:

```
del temp 
```

To enable the system to use Startup.new when booting, rename the original Startup file:

```
rename Startup Startup.old
```

Then rename Startup.new:

```
rename Startup.new Startup
```

OS9GEN

Syntax: `os9gen devname [opts]`

Function: Creates and links the required OS9Boot file to a diskette making it a bootable diskette.

Parameters:

devname The disk drive containing the diskette to receive the new boot file.

opts One or more of the following options.

Options:

-s Causes OS9GEN to use only one drive to generate the boot file. In a single-drive operation, OS9GEN reads the modules from the source diskette and asks you to exchange diskettes and press as it reads and copies the modules.

#*n*[K] reserves *n* kilobytes of memory for use by the OS9GEN command. By setting aside as much memory as possible, you can increase the speed of OS9GEN and, on single-drive systems, reduce the number of diskette swaps. If you type K after #*n*, the memory specified by *n* is in kilobytes (1024 bytes), otherwise *n* is in 256-byte pages.

Notes:

- OS9Boot files can only exist on contiguous sectors. Therefore, use OS9GEN only with newly formatted diskettes. If OS9Boot is fragmented, the system warns you not to use the diskette to bootstrap OS-9.

- OS9GEN creates a working file called Tempboot on the device specified by *devname*. Next, it reads filenames (path-lists) either from the keyboard (the standard input path) or redirected from a file. If you enter names manually, OS9GEN does not display a prompt. Type each filename and press **ENTER**. After typing the last filename and pressing **ENTER**, press **ENTER** again, or press **CTRL****BREAK** to complete the list.

OS9GEN opens each file and copies it to Tempboot. The process repeats until it reaches a blank line or an end-of-file marker. All of the modules listed in Chapter 5 are not required in a boot file. These modules must be included in a boot File:

OS9p2, Init, IOMan, RBF, SCF, CC3IO, VDGInt (or GrfInt), CC3Disk, D0, TERM, Clock, CC3G0.

- You must have RENAME in the current execution directory or in memory for OS9GEN to work properly.
- With all input files copied to Tempboot, OS9GEN deletes the OS9Boot file, if it exists. It renames Tempboot as OS9Boot, and writes the file's starting address and size in the diskette's Identification Sector (LSN 0) for use by the OS-9 bootstrap firmware. OS-9 writes its kernel on diskette Track 34. If there is not room for the kernel, an error message appears, and the operation terminates.
- If you have only one drive, you can generate a new boot file more easily using the CONFIG utility. CONFIG is designed to make custom system diskettes using either single- or multiple-drives.

Examples:

- The following commands manually install a boot file on device /D1 that is an exact copy of the OS9Boot file on device /D0. The first command line runs OS9GEN, the second enters the name of the file to install, and the third enters an end-of-file marker.

```
os9gen /d1 ENTER  
/d0/os9boot ENTER  
CTRLBREAK
```

- The following commands let you manually install a boot file on device /D1 that is a copy of the OS9Boot file on device /D0 and the modules stored in the files /D0/Tape.driver and /D2/Video.driver. Line 1 executes OS9GEN. Line 2 enters the main boot filename. Lines 3 and 4 enter the names of the two additional files, and Line 5 enters an end-of-file marker.

```
os9gen /d1   
/d0/os9boot   
/d0/tape.driver   
/d2/video.driver   
 
```

- The following commands generate a new boot file on Drive /D1 that includes all the old boot file modules. Line 1 uses BUILD to create a file called Bootlist. The next three lines enter the names of the three files into Bootlist. Line 5 terminates BUILD, and Line 6 runs OS9GEN with input redirected from the new Bootlist file.

```
build /d0/bootlist   
? /d0/os9boot   
? /d0/tape.driver   
? /d0/video.driver   
?   
os9gen /d1</d0/bootlist 
```

- To install a custom boot file on a single-drive system, build a Bootlist to drive the OS9GEN program. You need a directory that contains the required file managers, device drivers, descriptors, and other files for the boot file. For example, to make a new boot file containing only the /TERM, /D0, /D1, and /P devices, first build a Bootlist such as:

```
build /d0/bootlist   
? term_vdg.dt   
? p.dd   
? d0_35s.dd   
? d1_35s.dd   
? os9p2   
? Init   
? IOMan   
? RBF.mn   
? CC3Disk.dr   
? SCF.mn   
? CC3IO.dr   
? vdgint.io   
? printer.dr   
? clock.60hz   
? cc3go 
```

Then use OS9GEN to create the new boot file on a separate diskette by typing:

```
os9gen /d0 -s #25K </d0/bootlist 
```

This command causes OS9GEN to use only one drive, 25K of buffer space, and the filenames previously stored in the Bootlist file.

You can expand this basic bootlist file to include other standard OS-9 modules such as window device descriptors, other disk drivers, descriptors, and terminal or modem descriptors.

All of the standard bootlist modules are contained in the MODULES directory on the BASIC09/CONFIG diskette.

PROCS

Syntax: `procs [e]`

Function: Displays a list of the processes running on the system. PROCS automatically adjusts its output for 32- or 80-column displays.

Options:

`e` Causes PROCS to display the processes of all users.

Notes:

- Normally PROCS lists only processes having the user's ID. The list is a *snapshot* taken at the instant PROCS executes. Processes switch states rapidly, usually many times per second.
- PROCS shows the user and process ID numbers, priority, state (process status), memory size (in 256 byte pages), primary program module, and standard input path.
- PROCS adjusts its output for 80 or 32 columns.

Examples:

- Because PROCS automatically adjusts for either 32- or 80-column displays, the following command can produce either format:

```
procs e 
```

Following is a possible 32-column display of PROCS:

Id	Pid	User#	Pty	Age	Sta
Sigl	Mem	StPtr	Primary		
=====					
2	1	0	128	128	\$80
0	3	\$78E2	Shell		
3	6	0	128	128	\$80
0	16	\$74B2	Basic09		
4	2	0	128	128	\$80
0	6	\$05F3	Procs		
5	0	0	128	128	\$80
0	3	\$6FB2	Shell		
6	0	0	128	129	\$80
0	3	\$68E2	Shell		

Following is a possible 80-column display of PROCS:

Id	Pid	User Number	Pty	Age	Sts	Sigl	Siz	Ptr	Primary	Module

2	1	0	128	128	\$80	0	3	\$78E2	Shell	
3	6	0	128	128	\$80	0	16	\$74B2	Basic09	
4	5	0	128	128	\$80	0	3	\$72E2	Shell	
5	0	0	128	129	\$80	0	3	\$6FB2	Shell	
6	0	0	128	129	\$80	0	3	\$68E2	Shell	
7	4	0	128	128	\$80	0	6	\$05F3	Procs	

PWD PXD

Syntax: `pwd`
 `pxd`

Function: PWD shows the path from the ROOT directory to the current data directory. PXD shows the path from the ROOT directory to the current execution directory.

Notes:

- OS-9 keeps a current data directory and current execution directory for each process. Use PWD and PXD to show where your current data and execution directories are located on the disk or disks you are using.

Examples:

- The following example uses a full pathlist. CHD changes the current data directory to the MANUALS directory.

```
chd /d1/steve/textfiles/manuals 
```

To display the full path to the data directory, type:

```
pwd 
```

The screen displays the data directory path:

```
/D1/STEVE/TEXTFILES/MANUALS
```

- The following commands cause the current data directory to move up one level in the directory hierarchy and then display the data directory.

```
chd .. 
```

```
pwd 
```

```
/D1/STEVE/TEXTFILES
```

- The following commands change the current data directory to the parent directory and then display the current data directory.

```
chd .. 
```

```
pwd 
```

```
/D1/STEVE
```

- The following command displays the current execution directory, CMDS.

```
pxd 
```

```
/D0/CMDS
```

RENAME

Syntax: `rename pathlist filename`

Function: Gives the specified file or directory a new name.

Parameters:

- pathlist* The current name of the file or directory.
- filename* The new name.

Notes:

- You must have write permission for the file.

Examples:

- To change a file's name from Blue to Purple, type:
`rename blue purple [ENTER]`
- To rename a file in the USER9 directory of Drive /D3, type:
`rename /d3/user9/test temp [ENTER]`
- In the following example, DIR displays the names of the files in the current data directory. RENAME changes the filename Animals to Mammals. Another DIR command shows that RENAME has performed properly.

```
dir [ENTER]
```

The screen displays:

```
Directory of . 16:22:53
myfile      animals

rename animals mammals [ENTER]
dir [ENTER]
```

The screen now shows:

```
Directory of . 16:23:22
myfile      mammals
```

SETIME

Syntax: `setime [yy/mm/dd hh:mm[:ss]]`

Function: Sets the system date and time, and activates the real time clock.

Parameters:

<i>yy</i>	The year in a two-digit format (86 for 1986).
<i>mm</i>	The month in a one or two-digit format (01 or 1 for January, 12 for December).
<i>dd</i>	The day of the month in a one- or two-digit format, such as 21.
<i>hh</i>	The hour in a one- or two-digit, 24-hour format (15 for 3 p.m.).
<i>mm</i>	Minutes in a one- or two-digit format, such as 03, 5, or 55.
<i>ss</i>	Seconds in a one- or two-digit format, such as 04, 5, or 25.

Options:

Specifying seconds in the new time entry is optional.

Notes:

- You can include the date and time parameters. If you do not, SETIME asks you for them.
- Numbers are one- or two- decimal digits using the space, colon, semicolon, or slash as delimiters.
- The CC3go module starts the clock on system startup, so multitasking is possible without use of the SETIME utility.

- If you do not set the date and time when booting OS-9, the system cannot accurately update the “Last modified” date and time for files.

Examples:

- To set the date and time to August 15, 1986, 3:45 p.m., type:

```
setime 86,08,15,15,45 
```

- To set the same date using a slightly different but equally acceptable format, type:

```
setime 86/08/15 15/45/00 
```

SETPR

Syntax: `setpr procID number`

Function: Changes the CPU priority of a process. The priority of a process determines the CPU time allotted to it under multi-tasking conditions.

Parameters:

- | | |
|---------------|--|
| <i>procID</i> | The number of the process for which you want to change the priority. |
| <i>number</i> | The new priority number. |

Notes:

- The process priority number is a decimal number in the range 1 (lowest priority) to 255. If you need information about the process ID number and current priority, use PROCS.
- You can use SETPR only on processes that have your user number.
- SETPR does not appear in the CMDS directory because it is built into the shell.
- A Super User (User 0) can set any process priorities.

Examples:

- To set or change the priority of Process 8 to 250, type:

```
setpr 8 250 
```

- In the following commands PROCS displays process ID numbers and other information. Then, SETPR sets Process 3 to a priority of 255. The final command confirms the change.

```
procs 
```

Following is a sample screen display:

Id	User			Mem Stack				Primary Module	
	Pid	Number	Pty	Age	Sts	Signl	Siz		Ptr
2	1	0	128	128	000	0	3	\$78E2	Shell
3	6	0	128	128	000	0	16	\$74B2	Basic09
4	2	0	128	128	000	0	6	\$05F3	Procs
5	0	0	128	128	000	0	3	\$6FB2	Shell
6	0	0	128	129	000	0	3	\$68E2	Shell

```
setpr 3 255 
```

```
procs 
```

Id	User			Mem Stack				Primary Module	
	Pid	Number	Pty	Age	Sts	Signl	Siz		Ptr
2	1	0	128	128	000	0	3	\$78B2	Shell
3	6	0	255	128	000	0	16	\$74B2	Basic09
4	5	0	128	128	000	0	3	\$72E2	Shell
5	0	0	128	129	000	0	3	\$6FB2	Shell
6	0	0	128	129	000	0	3	\$68E2	Shell
7	4	0	128	128	000	0	6	\$05F3	Procs

SHELL

Syntax: `shell arglist`

Function: The shell is OS-9's command interpreter program. It reads data from its standard input path, processes it and sends the output to its standard output path, and sends error messages (and some prompts) via the standard error output. Any or all of these paths may be redirected. It interprets the data as a sequence of commands. The function of the shell is to initiate and control execution of other OS-9 programs.

Parameters:

arglist The commands, parameters, and options given SHELL in a command line.

Notes:

- The shell reads and interprets one text line at a time from the standard input path until it reaches an end-of-file marker. At that time it terminates itself.
- When another program calls the shell, a special case occurs in which the shell takes the argument list as its first line of input. If this command line consists of *built-in* commands only, the shell reads and processes more lines. Otherwise, control returns to the calling program after the shell processes the single command line.
- When operating from the shell, you do not need to specify the SHELL command to execute a program, a command, or a built-in shell function. Using SHELL before a command causes the existing shell to fork an additional shell, which then forks the specified process, such as:

```
shell dir e 
```

Issuing a command without SHELL causes the existing shell to fork the specified process, such as:

```
dir e 
```

The following two commands also have identical effects:

```
shell x ENTER
```

```
x ENTER
```

- The shell command separators are:
 - ;
 - &
 - !
 - ENTER end-of-line (sequential execution separator)
- The Shell command modifiers are:
 - < Redirect standard input
 - > Redirect standard output
 - >> Redirect standard error output
 - <> Redirects standard input and standard output
 - <>> Redirects standard input and standard error output
 - >>> Redirects standard output and standard error output
 - <>>> Redirects standard input, standard output and standard error output
 - #n Set the process memory size in pages
 - #nK Set the program memory size in 1 kilobyte units.
- The following built-in Shell command parameters tell OS-9 to:

<i>chd pathlist</i>	Change the data directory
<i>kill procID</i>	Send the termination signal to process
<i>setpr procID number</i>	Change the specified process priority

<code>chx pathlist</code>	Change the execution directory
<code>i = devicename</code>	Create an immortal process
<code>w</code>	Wait for any process to die
<code>p</code>	Turn on prompting
<code>-p</code>	Turn off prompting
<code>t</code>	Echo input lines to standard output
<code>-t</code>	Not echo input lines
<code>-x</code>	Not terminate on an error
<code>x</code>	Terminate on error
<code>*</code>	Not process the following text

- See Chapter 3 for more information on the operation of the shell.

TMODE

Syntax: `tmode` [*pathnum*] [*paramlist*] [...]

Function: Displays or changes the initialization parameters of the terminal. TMODE automatically adjusts its output for 32- or 80-column displays.

Common uses include changing baud rates and control key definitions.

Parameters:

pathnum One of the standard path numbers:
.0 = standard input path
.1 = standard output path
.2 = standard error output path

paramlist One of the following options.

Options:

`upc` Displays uppercase characters only. Lowercase characters automatically convert to uppercase.

`-upc` Displays both upper- and lowercase characters.

`bsb` Causes a backspace to erase characters. Backspace characters echo as a backspace-space-backspace sequence. This setting is the system default.

`-bsb` Causes backspace not to erase. Only a single backspace echoes.

`bsl` Enables *backspace over a line*. Deletes lines by sending backspace-space-backspace sequences to erase a line (for video terminals). This setting is the system default.

-bsl	Disables <i>backspace over a line</i> . To delete a line, TMODE prints a <i>new line</i> sequence (for hard-copy terminals).
echo	Input characters <i>echo</i> on the terminal. This setting is the system default.
-echo	Turns off the echo default.
lf	Turns on the auto line feed function. Line feeds automatically echo to the terminal on input and output carriage returns. The auto line feed setting is the system default.
-lf	Turns off the auto line feed default.
null = <i>n</i>	Sets the null count—the number of null (\$00) characters transmitted after carriage returns for the return delay. The value <i>n</i> is in decimal. The default is 0.
pause	Turns on the screen pause. This setting suspends output when the screen fills. See the <i>pag</i> parameter for a definition of screen size. Resume output by pressing the space bar. This setting is the system default.
-pause	Turns off the screen pause mode.
pag = <i>n</i>	Sets the length of the video display page to <i>n</i> (decimal) lines. This setting affects the <i>pause</i> mode.
bsp = <i>h</i>	Sets the backspace character for input. The value <i>h</i> is in hexadecimal. The default is 08.
del = <i>h</i>	Sets the delete line character for input. The value <i>h</i> is in hexadecimal. The default is 18.
eor = <i>h</i>	Sets the end-of-record (carriage return) character for input. This setting requires a value in hexadecimal. The default is 0D.
eof = <i>h</i>	Sets the end-of-file character for input. The value <i>h</i> is in hexadecimal. The default is 1B.

<code>reprint = h</code>	Sets the reprint line character. The value <i>h</i> is in hexadecimal.
<code>dup = h</code>	Sets the character to duplicate the last input line. The value <i>h</i> is in hexadecimal. The default is 01.
<code>psc = h</code>	Sets the pause character. The value of the character is in hexadecimal. The default is 17.
<code>abort = h</code>	Sets the terminate character (normally CONTROL C). The value of the character is in hexadecimal.
<code>quit = h</code>	Sets the quit character (normally CONTROL E). The value of the character is in hexadecimal.
<code>bse = h</code>	Sets the backspace character for output. The value <i>h</i> is in hexadecimal. The default is 08.
<code>bell = h</code>	Sets the bell (alert) character for output. The value <i>h</i> is in hexadecimal. The default is 07.
<code>type = h</code>	For external devices, use type for ACIA (asynchronous communications interface adapter) initialization values (hexadecimal). The default is 00. Bits 5-7 set either MARK, SPACE, or no parity on all devices. Codes for these are: 000 = no parity 101 = MARK parity transmitted, no checking 111 = SPACE parity transmitted, no checking 011 = even parity (available only with the external ACIA pak and Mod-pak devices) 001 = odd parity (available only with the external ACIA pak and Mod-pak devices)

Bit 4 selects auto-answer modem support features.

1 = on

0 = off

See “Technical Information for the RS232 Port” in Chapter 5 for more information.

For TERM-VDG, the type byte has a different use:

Bit 0 specifies a machine with true lowercase capability. Set Bit 0 to turn on true lowercase.

For TERM-WIN, use a value of 80 to specify a window device.

xon = *h* Sets the character to be used as a signal for resuming transmission of data after an xoff signal is received. Default is 0 (not active).

xoff = *h* Sets the character to be used for stopping data transmission. Default is 0 (not active).

baud = *h* Sets the baud rate, word length, and stop bits for a software-controllable interface. The codes for the baud rate are:

0 = 110 3 = 1200 6 = 9600

1 = 300 4 = 2400 7 = 19200 (ACIAPAK only)

2 = 600 5 = 4800 7 = 32000 (SIO only)

Bits 0-3 determine the baud rate.

Bit 4 is reserved for future use.

Bits 5-6 determine the word length:

00 = 8 bits

01 = 7 bits

Bit 7 determines the number of stop bits:

0 = 1 stop bit

1 = 2 stop bits

See “Technical Information for the RS232 Port” in Chapter 5 for further information.

Notes:

- You can specify any number of parameters from the options list, separating them by spaces or commas. If you don't specify parameters, TMODE displays the current values of the available options.
- You can use a period and a number to specify the pathnumber on which to read or set options. If you don't specify a path, TMODE affects the standard input path.
- TMODE works only if a path to the file/device is open. Use XMODE to alter device descriptors and set device initial operating parameters.
- TMODE can also alter the baud rate, word length, stop bits, and parity for devices already initialized.
- If you use TMODE in a procedure file, you must specify one of the standard output paths (.1 or .2). This procedure is necessary, because the command redirects the SHELL's standard input path to come from a disk file. (You can use TMODE only on SCFMAN-type devices.) For example, to set lines per page for standard output, use this line:

```
TMODE .1 pag=24 
```

Examples:

- The following command line sets the terminal to display upper- and lowercase, sets the null count to 4, and turns on the screen pause function.

```
tmode -upc lf null=4 pause 
```

- The next command sets the screen page length (number of lines) to 24, turns on the screen pause function and the backspace-over-line function, and sets the backspace character value to 8 and turns off the echo default.

```
tmode pag=24 pause bsl -echo bsp=8 
```

TUNEPORT

Syntax: `tuneport [device] [-s = value]`

Function: Lets you test and set delay loop values for the current baud rate and select the best value for your printer or terminal.

Parameters:

<i>device</i>	The device you want to test, either your printer (/p) or terminal (/t1).
<i>value</i>	A new delay loop value.

Options:

`-s =` Sets a new delay loop value.

Examples:

- The following command provides a test operation for your printer.

```
tuneport /p 
```

After a short delay, TUNEPORT displays the current baud rate and sends data to the printer to see if it is working properly. The program then displays the current delay value and asks for a new value. Enter a decimal delay value and press . Again, TUNEPORT sends data to the printer as a test. Continue this process until you find the best value. When you are satisfied, press instead of entering a value at the prompt. A closing message displays your new value.

Use the same process to set a new delay loop value for the /T1 terminal.

- The following command line sets the delay loop value for your printer to 255.

```
tuneport /p -s=255 
```

Use such a command on future system boots to set the optimum delay value determined with the TUNEPORT test function. Then, using OS9GEN or COBBLER, generate a new boot file for your system diskette. You can also use the -s option with TUNEPORT in your system Startup file to set the value.

UNLINK

Syntax: `unlink modname [...]`

Function: Tells OS-9 that the named memory module(s) is no longer needed by the user.

Parameters:

modname One or more modules you want to unlink.

Options:

In one command line, you can specify as many modules as you want to unlink.

Notes:

- Whether OS-9 destroys the modules and reassigns their memory depends on whether the module is in use by other processes. Each process using a module increases its link-count by one. Each UNLINK you issue decreases its link-count by 1. When the link-count reaches 0, OS-9 deallocates the module.
- You should unlink modules whenever possible to make most efficient use of available memory resources. Modules you have loaded and linked might need to be unlinked twice to remove them from memory.

- **Warning:** Never attempt to unlink a module you didn't load or link, and never unlink a module that is in use by programs (displayed by the PROCS command).

Examples:

- To unlink three modules named Pgm1, Pgm5, and Pgm99, type:

```
unlink pgm1 pgm5 pgm99 
```

- In the following command sequence, MDIR first displays the modules in memory. The next command unlinks the edit module. The output of the final command (MDIR) shows that UNLINK is successful—Edit no longer appears on the list.

```
mdir 
```

A possible screen display is:

```
Module Directory at 00:01:08
REL      Boot      DS9p1      DS9p2      Init
IDMan    RBF      CC3Disk    D0         D1
DD       SCF      CC3ID      VDGInt     GrfInt
TERM     W        W1         W2         W3
W4       W5       W6         W7         ACIAPAK
T2       PRINTER  P          PipeMan    Piper
Pipe     Clock    CC3Go     CC3HDisk   H0
Shell    Copy     Date      DEIniz     Del
Dir      Display  Echo      Iniz       Link
List     Load     MDir      Merge      Mfree
Procs    Rename   Setime    Tmode      Unlink
Basic09  GrfDrv   Edit
```

```
unlink edit 
mdir 
```

The new screen display is:

```
Module Directory at 00:03:15
REL      Boot      OS9p1      OS9p2      Init
IOMan    RBF      CC3Disk    D0         D1
DD       SCF      CC3IO      VDGInt     GrfInt
TERM     W        W1         W2         W3
W4       W5       W6         W7         ACIAPAK
T2       PRINTER  P          PipeMan    Piper
Pipe     Clock    CC3Go      CC3HDisk   H0
Shell    Copy     Date       Delniz     Del
Dir      Display  Echo       Iniz       Link
List     Load    MDir       Merge      Mfree
Procs    Rename   Setime     Tmode      Unlink
Basic09  GrfDrv
```

WCREATE

Syntax: `wcreate /wpath [-s = type] xpos ypos xsize ysize foreground background [border]`

Function: Initializes and creates a window.

Parameters:

<i>wpath</i>	The window device name of the window you are creating (W, W1, W2, W3, and so on).
<i>xpos</i>	The x co-ordinate (in decimal) for the starting position of the upper left corner of the screen.
<i>ypos</i>	The y co-ordinate (in decimal) for the starting position of the upper left corner of the screen.
<i>xsize</i>	The horizontal size of the screen in columns; 1 to 80 (in decimal) for screen types 2, 5, and 7, and 1 to 40 (decimal) for screen types 1, 6, and 8.
<i>ysize</i>	The vertical size of the screen in lines, in the range 1 to 24 (in decimal).
<i>foreground</i>	The window foreground color.
<i>background</i>	The window background color.
<i>border</i>	An optional window border color. The default is black.

Options:

-s = type The screen type, chosen from the following list:

Type	Description
1 =	40-column hardware text screen
2 =	80-column hardware text screen
5 =	640 x 192 two-color screen
6 =	320 x 192 four-color screen

7 = 640 x 192 four-color screen
8 = 320 x 192 sixteen-color screen

If you use the `-s=type` option, you must specify a border color in the command line. The `-s` option is only used to create a window on a new screen. When creating additional windows on the currently displayed screen, omit the `-s` and border color options.

- `-z` Directs WCREATE to accept input from the standard input (redirected from a file).
- `-?` Produces a help message for the command.

Examples:

- To create a full screen, 80-column text window on /w1, type:

```
wcreate /w1 -s=2 0 0 80 24 7 4 1 
```

- To create two windows (/w2 and /w3) on a 640 x 192 graphics screen in which /w2 is the upper left of the display and /w3 is the right half of the display, first use build to create an input file:

```
build wfile   
? /w2 -s=07 0 0 40 12 7 4 1   
? /w3 40 0 40 24 4 7   
? 
```

Then, create the windows using Wfile as input:

```
wcreate -z < wfile 
```

- You can use the `-z` option to create windows in your system startup file. For example, the following startup file sets up several windows, along with the usual `SETIME`.

```
* lock the shell in memory and set the time
  link shell
setime < /1

* create the new windows
wcreate -z
* set up an 80-column full window for /w1
/w1 -s=2 0 0 80 24 7 4 1
* set up a 40 column full window for /w2
/w2 -s=1 0 0 40 24 7 4 1
* set up /w3 and /w4 as halves of a
*640 x 192 display
/w3 -s=7 0 0 40 24 7 4 1
/w4 40 0 40 24 4 7
* the following blank line terminates input
* from wcreate

* get the graphics fonts loaded
merge sys/stdfonts > /w1
```

Now, when the system boots, it has four windows defined, besides `TERM`. As shown, you can use an asterisk as the first character on a line in order to allow comments in the file.

XMODE

Syntax: `xmode devname [paramlist]`

Function: Displays or changes the initialization parameters of any SCF-type device such as the video display, printer, RS-232 port, and others. XMODE automatically adjusts its output for 32- or 80-column displays.

Common uses include changing baud rates and control key definitions.

Parameters:

- pathnum* The device name to change, such as /term, /w7, /t2, and so on.
- paramlist* One of the following options.

Options:

- `upc` Displays uppercase characters only. Lowercase characters automatically convert to uppercase.
- `-upc` Displays both upper- and lowercase characters.
- `bsb` Causes a backspace to erase characters. Backspace characters echo as a backspace-space-backspace sequence. This setting is the system default.
- `-bsb` Causes backspace not to erase. Only a single backspace echoes.
- `bsl` Enables *backspace over a line*. Deletes lines by sending backspace-space-backspace sequences to erase a line (for video terminals). This setting is the system default.
- `-bsl` Disables *backspace over a line*. To delete a line, you must print a *new line* sequence (for hard-copy terminals).

echo	Input characters <i>echo</i> on the terminal. This setting is the system default.
-echo	Turns off the echo default.
lf	Turns on the auto line feed function. Line feeds automatically echo to the terminal on input, and they output carriage returns. The auto line feed setting is the system default.
-lf	Turns off the auto line feed default.
null = <i>n</i>	Sets the null count—the number of null (\$00) characters transmitted after carriage returns for the return delay. The value <i>n</i> is in decimal. The default is 0.
pause	Turns on the screen pause. This setting suspends output when the screen fills. See the <i>pag</i> parameter for a definition of screen size. Resume output by pressing the space bar. This setting is the system default.
-pause	Turns off the screen pause mode.
pag = <i>n</i>	Sets the length of the video display page to <i>n</i> (decimal) lines. This setting affects the <i>pause</i> mode.
bsp = <i>h</i>	Sets the backspace character for input. The value <i>h</i> is in hexadecimal. The default is 08.
del = <i>h</i>	Sets the delete line character for input. The value <i>h</i> is in hexadecimal. The default is 18.
eor = <i>h</i>	Sets the end-of-record (carriage return) character for input. This setting requires a value in hexadecimal. The default is 0D.
eof = <i>h</i>	Sets the end-of-file character for input. The value <i>h</i> is in hexadecimal. The default is 1B.
reprint = <i>h</i>	Sets the reprint line character. The value <i>h</i> is in hexadecimal.
dup = <i>h</i>	Sets the character to duplicate the last input line. The value <i>h</i> is in hexadecimal. The default is 01.

<code>psc = h</code>	Sets the pause character. The value of the character is in hexadecimal. The default is 17.
<code>abort = h</code>	Sets the terminate character (normally CONTROL C). The value of the character is in hexadecimal.
<code>quit = h</code>	Sets the quit character (normally CONTROL E). The value of the character is in hexadecimal.
<code>bse = h</code>	Sets the backspace character for output. The value <i>h</i> is in hexadecimal. The default is 08.
<code>bell = h</code>	Sets the bell (alert) character for output. The value <i>h</i> is in hexadecimal. The default is 07.
<code>type = h</code>	For external devices, use <code>type</code> for ACIA (asynchronous communications interface adapter) initialization values (hexadecimal). The default is 00. Bits 5-7 set either MARK, SPACE, or no parity on all devices. Codes for these are:

000 = no parity

101 = MARK parity transmitted, no checking

111 = SPACE parity transmitted, no checking

011 = even parity (available only with the external ACIA pak and Mod-pak devices)

001 = odd parity (available only with the external ACIA pak and Mod-pak devices)

Bit 4 selects auto-answer modem support features.

1 = on

0 = off

See "Technical Information for the RS232 Port" in Chapter 5 for more information.

Notes:

- XMODE is similar to TMODE, but there are differences. TMODE operates only on open paths, so its effect is temporary. XMODE updates the device descriptor. Its change persists as long as the computer is running, even if you or the system repeatedly open and close the paths to the device.
- If you use XMODE to change parameters and the COBBLER program to make a new system diskette or to remake the boot tracks on the current system diskette, the process permanently changes the parameters on the new system diskette.
- XMODE requires that you specify a device name. If you do not specify parameters, XMODE displays the present value for each parameter. You can use any number of parameters, separating them with spaces or commas.

Examples:

- The following command sets the term (video) for upper- and lowercase, the null count to 4, the backspace character value to 1F hexadecimal, and turns on the screen pause function.

```
xmode /term -upc null=4 bse=1F pause 
```

Macro Text Editor

Overview

The OS-9 Macro Text Editor is a powerful, easy-to-learn text-preparation system. Use it to prepare text for letters and documents or text to be used by other OS-9 programs, such as the assembler and high-level languages. The text editor includes the following features:

- Compact size
- Capability of having multiple read and write files open at the same time
- All OS-9 commands usable inside the text editor
- Adjustable workspace size
- Repeatable command sequences
- Edit macros (special utility functions)
- Multiple text buffers
- Powerful commands

The Macro Text Editor is about 5 kilobytes in size and requires at least 2K bytes of free RAM to run.

Text Buffers

As you enter text, the editor places it in a temporary storage area called a text buffer. A text buffer acts as a scratch pad for saving text that you can manipulate with various edit commands. The Macro Text Editor can use multiple text buffers, one at a time.

A buffer in use is called the *edit buffer*. Edit also has another default buffer called the *secondary buffer*. As well, you can create additional buffers up to the capacity of your computer's memory.

Edit Pointers

The Macro Text Editor has an edit *pointer* that identifies your position in the buffer, in a manner similar to holding your place in a book with your finger.

The pointer is invisible to you, but Edit commands can reposition it and display the text to which it points. Each buffer has its own edit pointer, and you can move from buffer to buffer without losing your place in any of them.

Entering Commands

The Macro Text Editor is interactive. This means you and the editor carry on a two-way conversation. You issue a command, and the editor carries out the command and displays the result. When you are through making changes, you can save your edited file, then press **Q** **ENTER** to quit editing.

When the editor displays **E:** on the screen, it is waiting for you to type a command. You type a line that includes one or more commands, then press **ENTER**. Edit carries out the commands and again displays **E:**.

If you enter more than one command on a line, separate the commands with a space. If, however, a space is the first character on a line, the editor considers the space to be an insert command and not a separator.

Correct a typing error by backspacing over it or by deleting the entire line. Note, you cannot correct a line after pressing **ENTER**.

Control Keys

You can use the same special control keys with Edit that you use with OS-9. See Appendix D for a complete listing of these keys. Following is a list of some of the control keys that are especially useful with Edit:

Control Key(s)	Function
CTRL A	Repeats the previous input line.
CTRL C	Terminates the editor and returns to command entry mode.
CTRL D	Displays the current input on the next line.
CTRL H or ←	Backspaces and erases the previous character.

Control Key(s)	Function
<input type="checkbox"/> Q <input type="checkbox"/> ENTER	Interrupts the editor and returns to command entry mode.
<input type="checkbox"/> CTRL <input type="checkbox"/> W	Temporarily halts the data output to your terminal so that you can read the screen before the data scrolls off. Output resumes when you press any other key.
<input type="checkbox"/> CTRL <input type="checkbox"/> X or <input type="checkbox"/> SHIFT <input type="checkbox"/> ←	Deletes the line.
<input type="checkbox"/> CTRL <input type="checkbox"/> BREAK	Terminates the editor, and returns to command entry mode.

Command Parameters

There are two types of edit parameters, “numeric” and “string.”

Numeric Parameters. Numeric parameters specify an amount, such as the number of times to repeat a command or the number of lines affected by a command. If you do not specify a numeric parameter, the editor uses the default value of one. Specify all other numeric parameters in one of the following ways.

- Enter a positive decimal integer in the range 0 to 65,535. For example:
0
10
5250
65532
31
- Enter an asterisk (*) as a shorthand for *all* (all the way to the beginning, all the way to the end, all of the lines, and so on). To the editor, an asterisk means infinity. Use the asterisk to specify all remaining lines, all characters, or repeat forever.
- Use a numeric variable. (See “Parameter Passing” later in this chapter.)

String Parameters. String parameters specify a single character, group of characters, word, or phrase. Specify string parameters in either of the following ways.

- Enclose the group of characters with delimiters (two matching characters). You can use any characters, but they must match. If one string immediately follows another, separate the two with a single delimiter that matches the others. For example:

```
"string of characters"  
/STRING/  
: my name is Larry :  
"first string"second string"  
/string 1/ string 2/
```

- Use a string variable. (See “Using Macros” later in this chapter.)

Syntax Notation

Syntax descriptions indicate what to enter and the order in which to do it. The command name is first; type it exactly as shown. Follow the command name with the correct parameters. Enter each as it is described in the section on parameters.

The syntax descriptions for each command use the following notations:

n = numeric parameter

str = string parameter

␣ = space character. When you see ␣, press the space bar.

text = one or more characters terminated by pressing

Getting Started

From the OS-9 prompt, start Edit by typing:

```
edit 
```

Enter a command when the screen shows E:.

You can quit Edit at any time by pressing . The Q command terminates the editor and returns you to the OS-9 Shell, which responds with the OS9: prompt.

Following is a list of ways you can start the editor, including the effect of each. The examples call a file that already exists *oldfile*. They call a file to be created *newfile*.

EDIT OS-9 loads the editor and starts it. The command does not establish an initial read or write files, but you can perform text file operations by opening files after the editor is started.

EDIT *newfile* OS-9 loads the editor and starts it, creating the file called *newfile*. *Newfile* is the initial write file. There is no initial read file. However, you can open files to read later.

EDIT *oldfile* OS-9 loads the editor and starts it. The initial read file is *oldfile*. The editor creates a file called SCRATCH as the initial write file. When you end the edit session, OS-9 deletes *oldfile* and renames SCRATCH to *oldfile*. This gives the appearance of *oldfile* being updated.

Note: The two OS-9 utilities DEL and RENAME must be present on your system if you wish to start the editor in this manner.

**EDIT *oldfile*
*newfile*** OS-9 loads the editor and starts it. The initial read file is *oldfile*. The editor creates *newfile*—the initial write file. The terms *oldfile* and *newfile* refer to any properly constructed OS-9 pathlist.

Edit Commands

Displaying Text

L*n* Lists (displays) the next *n* lines, starting at the current position of the edit pointer. The edit pointer position does not change.

1

displays the current line. If the edit pointer is not at the beginning of the line, only the portion of the line to the right of the edit pointer shows on the screen.

13

displays the current line and the next two lines.

1 *

displays all text from the current position of the edit pointer to the end of the buffer.

The L command displays text regardless of which verify mode is in effect.

X*n* Displays the *n* lines that precede the edit pointer. The position of the edit pointer does not change. For example:

x

displays any text on the current line that precedes the edit pointer. If the edit pointer is at the beginning of the line, the command displays nothing.

x3

displays the two preceding lines and any text on the current line that precedes the edit pointer.

The X command displays text regardless of which verify mode is in effect.

Manipulating the Edit Pointer

CTRL **7** or **↑** Moves the edit pointer to the beginning (first character) of the text buffer. The screen shows the up arrow when you hold down **CTRL** and press **7**. For example,

CTRL **7** **ENTER**

moves the edit pointer to the beginning of the buffer.

/ Moves the edit pointer to the end (last character) of the buffer. For example,

/ **ENTER**

moves the edit pointer past the end of the buffer.

ENTER Moves the edit pointer to the beginning of the next line and displays it. Use this command to go through text one line at a time. You can look at each line, correct any mistakes, and then move to the next line.

+ n

Moves the edit pointer either to the end of the line or forward *n* lines and displays the line. Entering a value of zero moves the edit pointer to the end of the current line. For example:

+ 0

Entering a value other than zero moves the pointer forward *n* lines and displays the line. For example,

+

moves the pointer to the next line and displays the line. This command performs the same function as .

+ 10

moves the pointer ahead 10 lines and displays the line.

+ *

moves the edit pointer to the end of the buffer.

- n

Moves the edit pointer either to the beginning of the line or backward *n* lines. For example:

- 0

moves the edit pointer to the beginning of the line and displays the line. Entering a value other than zero moves the edit pointer back *n* lines. For example,

-

moves the edit pointer back one line and displays the line.

- 5

moves the edit pointer back five lines and displays the line.

- *

moves the edit pointer to the beginning (top) of the buffer and displays the first line.

>n

Moves the edit pointer to the right *n* characters. Use this command to move the edit pointer to some position in the line other than the first character. For example,

>

moves the edit pointer to the right one character.

>25

moves the edit pointer to the right 25 characters.

>*

moves the edit pointer to the end of the buffer.

<n

Moves the edit pointer to the left *n* characters. Use this command to move the edit pointer to some position in a line other than the first character. For example:

<

moves the edit pointer to the left one character.

<10

moves the edit pointer to the left 10 characters.

<*

moves the edit pointer to the beginning of the buffer.

Inserting and Deleting Lines

***I*text** Preceding text lines with a space inserts the text as a new line ahead of the edit pointer. The position of the edit pointer does not change. For example,

```
  IInsert this line 
```

inserts the line.

```
  ILine one   
  ILine two   
  ILine three 
```

inserts three lines.

***I*n str** Inserts a line of *n* copies of the specified string immediately before the position of the edit pointer. The position of the edit pointer does not change. For example,

```
  i40/*/ 
```

inserts a line containing 40 asterisks. You can also use the “I” command to insert a line containing a single copy of the string. This function is important when you want to use a macro to insert lines, since the space bar cannot be used within a macro. For example,

```
  i"Line to insert" 
```

inserts the line.

Dn

Deletes (removes) *n* lines from the edit buffer, starting with the current line. This command displays the lines to be deleted. For example:

d

deletes the current line, regardless of the position of the edit pointer, and displays it.

d4

deletes the current line and the next three lines.

d*

deletes everything from the current line to the end of the buffer.

Kn

Kills (deletes) *n* characters, starting at the current position of the edit pointer. This command displays all deleted characters. For example,

k

deletes the character at the edit pointer.

k4

deletes the character at the current position of the edit pointer and the next three characters.

k*

deletes everything from the current position of the edit pointer to the end of the buffer.

En str Extends *n* lines by adding a string to the end of each line. E extends a line, displays it, and then moves the pointer past it. For example,

```
e/this is a comment/ 
```

adds the string “this is a comment” to the end of the current line and moves the edit pointer to the next line.

```
e3/xx 
```

adds the string xx to the end of the current line and the next two lines. It moves the pointer past these lines.

U Unextends (deletes) the remainder of a line from the current position of the pointer. Use U to remove extensions, such as comments, from a line. For example,

```
u 
```

deletes all the characters from the current position of the pointer up to the end of the current line.

For some practice in using the commands that display text, manipulate the edit pointer, and insert and delete lines, turn to Sample Session 1 in this chapter.

Searching and Substituting

Sn string

Searches for the next *n* occurrences of *string*. When Edit finds an occurrence, it displays the line and moves the edit pointer to the line. If Edit does not find the string or if all the occurrences have been found, the edit pointer does not move. For example,

```
s/my string/ 
```

searches for the next occurrence of “my string”.

```
s3"strung out" 
```

searches for the next three occurrences of “strung out”.

```
s*/seek and find/ 
```

searches for all occurrences of “seek and find” between the edit pointer and the end of the text.

Cn string1 string2

Changes the next *n* occurrences of *string1* to *string2*. When Edit finds *string1*, it moves the edit pointer past it and changes *string1* to *string2*, then it displays the updated line. If it does not find *string1* it displays “NOT FOUND.” If all the occurrences have been found, the edit pointer does not move. For example,

```
c/this/that/ 
```

changes the next occurrence of “this” to “that”.

```
c2/in/out/ 
```

changes the next two occurrences of “in” to “out”.

```
c!*seek and find!sought and  
found! 
```

changes all occurrences of “seek and find” that are between the edit pointer and the end of text to “sought and found”.

An Sets the SEARCH/CHANGE anchor to Column *n*. To find a string that begins in a specific column, set the anchor to the column position before using the search command to find it. If you do not include a value for *n*, Edit assumes Column 1. For example:

a

finds a string only if it begins in Column 1.

a20

finds a string only if it begins in Column 20. If you use the A command to set the anchor, this setting remains in effect only for the current command line. After Edit executes the command, the anchor automatically returns to its normal value of zero.

For some practice in using the commands that search and substitute, turn to Sample Session 2 in this chapter.

Miscellaneous Commands

Tn Tabs (moves) the edit pointer to Column *n* of the current line. If *n* exceeds the line length, this command extends the line with spaces. For example,

t

moves the edit pointer to Column 1 of the current line.

t5

moves the edit pointer to Column 5 of the current line.

**.SHELL
command
line**

Lets you use any OS-9 command from within the editor. The remainder of the command line following .SHELL passes to the OS-9 Shell for execution. For example,

```
.shell dir /d1 
```

calls the OS-9 Shell to display the directory of D1.

```
.shell basic09 
```

starts BASIC09.

```
.shell edit oldfile newfile 
```

restarts the editor.

Mn

Adjusts the amount of memory available for buffers and macros. If the workspace is full and the editor does not allow you to enter more text, increase the workspace size. If you need only a small amount of the available workspace, decrease the workspace size so that other OS-9 programs can use the memory. For example,

```
m5000 
```

sets the workspace size to 5000 bytes.

```
m10000 
```

sets the workspace size to 10000 bytes.

Before leaving Edit, you can increase the workspace. This decreases the time the editor takes to copy the input file to the output file, because the editor can read and write more data at one time. Edit changes memory in 256-byte pages. For the M command to have any effect, a new workspace size must differ from the current size by at least 256 bytes. The M command does not let you deallocate any workspace that Edit needs for buffers or macros.

.SIZE Displays the size of the workspace and the amount that has been used. For example:

```
.size
521    15328
```

521 is the amount of workspace Edit uses for buffers and macros. 15328 is the amount of available memory.

Q Ends editing and returns to the OS-9 Shell. If you specified files when you started, Edit writes the text in Buffer 1 to the initial write file (specified when you start Edit). Next it copies the remainder of the initial input file (specified when you start Edit) to the initial write file. The editor then terminates, and control returns to the OS-9 Shell.

Vmode Turns the verify mode on or off. Edit always starts with the verify mode on. Therefore, the editor displays the results of all the commands for which verify is appropriate. If you do not want to see the results of commands, turn off the verify mode by specifying 0 (zero) for *mode*. To turn verify back on, specify any non-zero number. For example,

```
v0 
```

turns off the verify mode.

```
v2 
```

turns on the verify mode.

```
v13 
```

turns on the verify mode.

If the verify mode is on when you switch to a macro, it remains on. If you turn off verify while in the macro, it is restored when you return to the editor.

Manipulating Multiple Buffers

.DIR Displays the directory of the editor's buffers and macros. For example:

```

BUFFERS:
$      0 (secondary buffer)
*      1 (primary buffer)
       5 (another buffer)

```

```

MACROS:

MYMACRO
LIST
COPY

```

Bn Makes buffer *n* the primary buffer. When you switch from one buffer to another, the old one becomes the secondary buffer, and the new one becomes the primary buffer. For example,

```
b5 
```

makes Buffer 5 the primary buffer. If Buffer 5 does not exist, Edit creates it.

Pn Puts (moves) *n* lines into the secondary buffer. This command removes the lines from the primary buffer, starting at the position of the edit pointer, and inserts them into the secondary buffer before the current position of the edit pointer. It displays the text that is moved. For example,

```
p 
```

moves one line to the secondary buffer.

```
p5 
```

moves five lines to the secondary buffer.

```
p* 
```

moves all lines that are between the current position of the edit pointer and the end of text to the secondary buffer.

Gn Gets (moves) *n* lines from the secondary buffer. This command takes the lines from the top of the secondary buffer and inserts them into the primary buffer before the current position of the edit pointer. Edit then displays the moved lines. When used with the P command, G moves text from one place to another. For example,

g

gets one line from the secondary buffer.

g5

gets five lines from the secondary buffer.

g*

gets all lines from the secondary buffer.

For some practice in using miscellaneous commands and the commands that manipulate multiple buffers, turn to Sample Session 3 in this chapter.

Text File Operations

This section of the manual describes the group of commands related to reading and writing OS-9 text files.

.NEW Gets new text. Use .NEW when editing a file that is too large to fit into the editor's workspace. .NEW writes out all lines that precede the current line, then appends an equal amount of new text to the end of the buffer.

.NEW always writes text to the initial output file (created when you start the editor) and always reads text from the initial input file (specified when you start the editor).

If you have finished editing the text currently in the buffer, you can "flush" this text and fill the buffer with new text by moving the edit pointer to the bottom of the buffer and then using the .NEW command. For example:

/ .new

If you wish to retain part of the text that is already in the buffer, move the edit pointer to the first line you wish to retain and then type `.new`. This command “flushes” all lines that precede the edit pointer. It then tries to read in new text that is the same size as the portion flushed out.

.READ *str*

Prepares an OS-9 text file for reading. *str* specifies the pathlist. For example.

```
.read "myfile" 
```

closes the current input file and opens “myfile” for reading.

You can specify an empty pathlist. For example,

```
.read "" 
```

closes the current input file and restores the initial input file (specified when you start the editor) for reading.

An open file remains attached to the primary buffer until you close the file. You can have more than one input file open at any time by using the `.READ` command to open them in different buffers.

To read these files, switch to the proper buffer, and then use the `R` command to read from that buffer’s input file. To close a file, you must be in the same buffer where the file was opened.

.WRITE *str* Opens a new file for writing. The *string* specifies the pathlist for the file you wish to create. For example,

```
.write "newfile" 
```

closes the current write file and creates one called "newfile". You can specify an empty pathlist. For example:

```
.write "" 
```

closes the current write file and restores the initial write file (specified when you start the editor).

.WRITE attaches a new write file to the primary buffer that remains attached until you close the file. You can have more than one write file open by using .WRITE to open them in different buffers. To write these files, switch to the proper buffer. To close a file, you must be in the same buffer where the file was opened.

R*n* Reads (gets) *n* lines of text from the buffer's input file. It displays the lines and inserts them before the current position of the edit pointer. For example,

```
r 
```

reads one line from the input file.

```
r 10 
```

reads 10 lines from the input file.

```
r * 
```

reads the remaining lines from the input file.

If a file contains no more text, the screen shows the *END OF FILE* message.

Wn Writes *n* lines to the output file, starting with the current line. It displays all lines that are deleted from the buffer. For example,

w

writes the current line to the output file.

w5

writes the current line and the next four lines to the output file.

w*

writes all lines from the current line to the end of the buffer to the output file.

For some practice in using the commands that read and write OS-9 text files, turn to Sample Session 4 in this chapter.

Conditionals and Command Series Repetition

When a command cannot be executed, the editor sets an internal flag, and the screen shows *FAIL*. For example, if you try to read from a file that has no more text, the editor sets the fail flag. A set fail flag means that the editor cannot execute any more commands until Edit encounters one of the following:

- The end of a command line typed from the keyboard.
- The end of the current loop. Any loops that are more deeply nested are skipped. (See the repeat command.)
- A colon (:) command. Since loops nested deeper than the current level are skipped, any occurrences of : that are in a more deeply nested loop are also skipped.

Following are the commands and conditions that set the fail flag:

- < Trying to move the edit pointer beyond the beginning of the edit buffer.
- > Trying to move the edit pointer beyond the + end of the buffer.
- S,C Not finding a string that was searched for.
- G No text left in the secondary buffer.
- R No text left in the read file.
- P,W No text left in the primary buffer.

If you specify an asterisk for the repeat count on these commands, Edit does not set the fail flag, because an asterisk usually means continue until there is nothing more to do. The following commands explicitly set the fail flag if some condition is not true.

- .EOF** Tests for end-of-file. .EOF succeeds if there is no more text to read from a file. Otherwise, it sets the fail flag.
- .NEOF** Tests for not end-of-file. .NEOF succeeds if there is text to read from the file. Otherwise, it sets the fail flag.
- .EOB** Tests for end-of-buffer. .EOB succeeds if the edit pointer is at the end of the buffer. Otherwise, it sets the fail flag.
- .NEOB** Tests for not end-of-buffer. .NEOB succeeds if the edit pointer is not at the end of the buffer. Otherwise, it sets the fail flag.
- .EOL** Tests for end-of-line. This test succeeds if the edit pointer is at the end of the line. Otherwise, it sets the fail flag.
- .NEOL** Tests for not end-of-line. .NEOL succeeds if the edit pointer is not at the end of the line. Otherwise, it sets the fail flag.
- .ZERO *n*** Tests for zero value. .ZERO succeeds if *n* equals zero. Otherwise, it sets the fail flag.

- .STAR *n*** Tests for star (asterisk). .STAR succeeds if *n* equals 65,535 (“*”). Otherwise, it sets the fail flag.
- .STR *str*** Tests for string match. .STR succeeds if the characters at the current position of the edit pointer match the string. Otherwise, it sets the fail flag.
- .NSTR *str*** Tests for string mismatch. .NSTR succeeds if the characters at the current position of the edit pointer do not match the string. Otherwise, it sets the fail flag.
- .S** Exits and succeeds. This is an unconditional exit from the innermost loop or macro. The fail flag clears after the exit.
- .F** Exits and fails. This is an unconditional exit from the innermost loop or macro. The fail flag sets after the exit.
- [*commands*]n** Repeats the *commands* *n* times. Left and right brackets form a loop that repeats the enclosed *commands* *n* times. (The loop must be repeated at least once.) If you enter the loop command from the keyboard, it must all be on one line. If it is part of a macro, however, it can span several command lines. For example,

```
[ ] 5 ENTER
```

repeats the L command five times.

Note: This is not the same as L5, which executes the L command only once and has 5 as its parameter.

- [+]*** Displays lines starting with the next line up to the end of the buffer and moves the edit pointer to the end of the buffer.

This command repeats until the operation reaches the end of the buffer. Then, when the command tries to move the edit pointer past the end of the buffer, Edit sets the fail flag, terminates the loop, then clears the fail flag.

: commands Executes the commands following the colon based on the state of the fail flag. For example:

FAIL FLAG CLEAR Skips all commands that follow the colon (:)
up to the end of the current loop or macro.

FAIL FLAG SET Clears the fail flag, and executes the commands that follow the colon (:).

Below is a command line that deletes all lines that do not begin with the letter A.

```
CTRL [7] [ .neob [ .str"A" + : d ]  
] * ENTER
```

^ moves the edit pointer to the beginning of the buffer. The outer loop tests for the end of the buffer and terminates the loop when it is reached.

The inner loop tests for A at the beginning of the line. If there is an A, the + command is executed. Otherwise, it executes the D command.

Below is a command that searches the current line for "find it". If the command finds the text, it displays the line. Otherwise, the command line fails and the screen shows * FAIL *.

```
[ .eol v0 -0 v .f : .str"find it"  
-0 .s : [ > ] ] * ENTER
```

.EOL V0 -0 V .F tests to determine if the edit pointer is at the end of the line. If it is, Edit turns off the verify mode to prevent -0 from displaying the line. Then it turns verify back on, and .F ends the loop.

If the edit pointer is not at the end of the line, the .STR command searches for “find it” at the current position of the edit pointer. If it is at the end of the line, Edit executes the -0 .S commands. This execution moves the edit pointer back to the beginning of the line, displays the line, and terminates the loop. Otherwise, the > command moves the edit pointer to the next position in the line.

The brackets prevent the command from failing and terminating the main loop if the end of the buffer is reached.

Edit Macros

Edit macros are commands you create to perform a specialized or complex task. For example, you can replace a frequently used series of commands with a single macro. First, save the series in a macro. Then each time you need it, type a period followed by the macro’s name and parameters. The editor responds as if you had typed the series of commands.

Macros consist of two main parts, the header and the body. The header gives the macro a name and describes the type and order of its parameters. The body consists of any number of ordinary commands. (Except for a space character and `[ENTER]`, you can use any command in a macro).

Note: Macros cannot create new macros.

To create a macro, first define it with the .MAC command. Then enter the header and body in the same manner as you enter text into an edit buffer. When you are satisfied with the macro, close its definition by pressing `[Q] [ENTER]`. This command returns you to the normal edit mode.

Macro Headers. A macro header must be the first line in each macro. It consists of a name, and a “variable list” that describes the macro’s parameters, if there are any. The name consists of any number of consecutive letters and underline characters. Following are possible macro names:

```
del_all
trim_spaces
LIST
CHANGE_X_TO_Y
```

Although you can make a macro name any length, it is better to keep it short, because you must spell it the same way each time you use it. You can use upper- and lowercase letters or a mixture.

Using Macros. Like other commands, you can give parameters to macros so that they are able to work with different strings and with different numbers of items. Macros are unable to use parameters directly. Instead, Edit passes the parameters on to the commands that make up the macro.

To pass the macro's parameters to these commands, use the variable list in the macro header to tell each command which of the macro's parameters to use. Each variable in the variable list represents the value of the macro parameter in its corresponding position. Use the corresponding variable wherever the parameter's value is needed.

The two types of variables are numeric and string. A numeric variable is a variable name preceded by the # character. A string variable is a variable name preceded by a \$ character. Variable names, like macro names, are composed of any number of consecutive letters and underline characters. Examples of numeric variables are:

```
#N
#ABC
#LONG_NUMBER_VARIABLE
```

Examples of string variables are:

```
$A
$B
$STR
$STR_A
$lower_case_variable_name
```

The function of the edit macro below is the same as that of the S command, to search for the next *n* occurrences of a string.

The first line of the macro is the macro header. It assigns the macro's name as SRCH. It also specifies that the macro needs one numeric parameter (#N) and one string parameter (\$STR). The entire body of the macro is the second line. This example passes both of the macro's parameters to the S command, which does the actual searching.

```
SRCH #N $STR
S #N $STR
```

Here is an example of how to execute this macro:

```
.SRCH 15 "string" 
```

In the next example, the order of the parameter is reversed. Therefore, when executing the macro, use the reverse order. The macro structure is:

```
SRCH $STR #N
S #N $STR
```

Specify the parameters for the "S" command in the proper order since it is only the "SRCH" macro that is changed. The following example shows how to execute this macro. The order of the parameters corresponds directly to the order of the variables in the variable list.

```
.SRCH "string" 15 
```

Macro Commands

Although macro editing has the same functions as text editing, the macro mode also includes some special commands. The macro commands you can use are as follows:

! *text* Places comments inside a macro. Ignores the remainder of the line following the ! command. This command lets you include, as part of a macro, a short description of what it does. Comments can help you remember the function of a macro. For example:

```
!  
<^>! Move the pointer to the top of the  
buffer.  
!*! Display all lines of text.  
!
```

In this example there are four comments. Two are empty, and two describe the commands that precede them.

.*macro name* Executes the macro specified by the name following the period (.). For example:

```
.mymacro   
.list 0   
.trim " "   
.merge " file_a " file b b" 
```

.MAC *str* Creates a new macro or opens the definition of an existing one so that it can be edited. To create a new macro, specify an empty string. For example,

```
.mac "" 
```

creates a new macro and puts you into the macro mode.

The screen shows M: instead of E: when the editor is in the macro mode. To edit a macro that already exists, specify the macro's name. For example,

```
.mac "mymacro" 
```

opens the macro "MYMACRO" for editing.

When a macro is open, edit it, or enter its definition with the same commands you use in a text buffer. After you edit the macro, press to close its definition and return to the edit mode. The first line of the macro must begin with a name that is not already used in order to close the definition and return to Edit.

**.SAVE *str1*
*str2***

Saves macros on an OS-9 file. *Str1* specifies a list of macros to be saved. Separate the macro names with spaces. *Str2* specifies the pathlist for the file on which you want to save the macros. For example:

```
.save "mymacro"myfile" 
```

saves the macro "MYMACRO" on the file "MYFILE".

```
.save "maca macb macc"mfile" 
```

saves the macros "MACA," "MACB," and "MACC" on the file "MFILE".

**.SEARCH *n*
*str***

Searches the text file buffer for the specified string. When a match is found, it stops and displays that line. The *n* option permits a search for the *nth* occurrence of a string match. This command is the same as S *n str*.

.LOAD *str* Loads macros from an OS-9 file. As each macro loads, Edit verifies that no other macro already exists with the same name. If one does, the macro with the duplicate name does not load, and Edit skips to the next macro on the file. Edit displays the names of all macros it loads. For example,

```
.load "macrofile" 
```

loads the macros in the file called MACROFILE.

```
.load "myfile" 
```

loads the macros in the file called MYFILE.

.DEL *str* Deletes the macro specified by the string. For example,

```
.del "mymacro" 
```

deletes the macro called MYMACRO.

```
.del "list" 
```

deletes the macro called LIST.

.DIR Displays the current edit buffer area. All edit buffers and macros currently in memory are displayed.

.CHANGE *n str1 str2* Changes the occurrence of *str1* to *str2*. The *n* option permits *n* occurrences of *str1* to be changed to *str2*.

Q Ends a macro edit session and returns you to the normal edit mode. For example:

```
Search_and_Delete #N $STR
!This example MACRO is used to
!check
!the string at the beginning of
!an #N number of lines. If the
!string matches, it will delete
!that line from the text buffer
!file.
!
!NOTE: The way the editor
!processes a MACRO causes it to
!see any parameters in the outer
!loop first. Thus, the #N
!parameter is processed before
!the STR parameter.
!
[^]          !Move to start of
             !edit buffer
[           !start of outer loop
.neob       !test for buffer end
[           !start of inner loop
.nstr $str  !test for not string
             !match
+           !go to next line if
             !no match
:           !if flag clear skip
             !next command
D           !delete line if flag
             !set
]           !end of inner loop
]#N        !end of outer loop
! End of Macro
```

For practice in using macro commands, turn to Sample Session 5 in this chapter.

Sample Session 1

Clear the buffer by deleting its contents.

You Type: D*

Screen Shows: ^D*

Insert three lines into the buffer. Begin each line with a space, which is the command for inserting text.

You Type:

Screen Shows: MY FIRST LINE
MY SECOND LINE
MY THIRD LINE

Move the edit pointer to the top of the text. The editor always considers the first character you type a command.

Note: always shows ^ on the screen. Typing - * also moves the edit pointer to the beginning of a buffer.

You Type:

Screen Shows: ^

List (display) the first line you inserted into the buffer.

You Type: L

Screen Shows: L
MY FIRST LINE

Display the first two lines you inserted into the buffer.

You Type: L2

Screen Shows: L2
MY FIRST LINE
MY SECOND LINE

Move to the next line and display it.

You Type:

Screen Shows: MY SECOND LINE

Move to the next line and display it.

You Type:

Screen Shows: MY THIRD LINE

Using L, display text beginning at the position of the edit pointer.

You Type: L
Screen Shows: L
 MY THIRD LINE

Insert a line into the buffer.

Note: In the next sample you see that the insert comes before the current position of the edit pointer.

You Type: INSERT A LINE
Screen Shows: INSERT A LINE

The following command line consists of more than one command. moves the edit pointer to the top of the text. L displays the text, and the asterisk (*) following L indicates that text is displayed through to the end of the buffer.

You Type: L *
Screen Shows: ^L *
 MY FIRST LINE
 MY SECOND LINE
 INSERT A LINE
 MY THIRD LINE

Show the position of the edit pointer.

You Type: L
Screen Shows: L
 MY FIRST LINE

Move the edit pointer forward two lines and display the lines.

You Type: +2
Screen Shows: +2
 INSERT A LINE

Display all lines from the edit pointer to the end of the buffer.

You Type: L *
Screen Shows: L *
 INSERT A LINE
 MY THIRD LINE

Move the edit pointer to the end of the buffer.

You Type: /
Screen Shows: /

Determine if the edit pointer is at the end of text. Since the screen shows no more lines, the edit pointer is at the end-of-text.

You Type: L *
Screen Shows: L *

Insert two more lines.

You Type: FIFTH LINE
 LAST LINE
Screen Shows: FIFTH LINE
 LAST LINE

Move the edit pointer back one line, and display the line.

You Type: -2
Screen Shows: -2
 FIFTH LINE

Move the edit pointer back two lines, and display the line.

You Type: -3
Screen Shows: -3
 MY SECOND LINE

Move the edit pointer three characters to the right and display the remainder of the line.

Note: You must put spaces between commands.

You Type: >3 L
Screen Shows: >3 L
 SECOND LINE

Display the characters that precede the edit pointer on the current line.

You Type: X
Screen Shows: X
 MY

Move the edit pointer to the end of the current line.

You Type: +
Screen Shows: +

Determine if the edit pointer is at the end of the line. It is, since the screen shows no lines.

You Type: L
Screen Shows: L

Display the characters that precede the edit pointer on the current line.

You Type: X
Screen Shows: X
 MY SECOND LINE

Move the edit pointer back to the beginning of the current line.

You Type: - 0
Screen Shows: - 0
 MY SECOND LINE

Determine if the edit pointer is at the beginning of the line. Since the screen shows no lines, the pointer is at the beginning.

You Type: X
Screen Shows: X

Go to the beginning of the text.

You Type:
Screen Shows: ^

Insert a line of 14 asterisks.

You Type: I 14 "*"
Screen Shows: I 14 "*"

Insert an empty line.

You Type: I ""
Screen Shows: I ""

Move to the top of the text, and display all lines in the buffer.

You Type: L *
Screen Shows: ^L *

 MY FIRST LINE
 MY SECOND LINE
 INSERT A LINE
 MY THIRD LINE
 FIFTH LINE
 LAST LINE

Move the edit pointer forward two lines.

You Type: +2
Screen Shows: +2
 MY FIRST LINE

Extend the line with XXX.

You Type: E" XXX"
Screen Shows: E" XXX"
 MY FIRST LINE XXX

Display the current line.

Note: The previous E command moved the edit pointer to the next line.

You Type: L
Screen Shows: L
MY SECOND LINE

Extend three lines with YYY.

You Type: E3"]YYY"
Screen Shows: E3" YYY"
MY SECOND LINE YYY
INSERT A LINE YYY
MY THIRD LINE YYY

Move back 2 lines.

You Type: -2
Screen Shows: -2
INSERT A LINE YYY

Move the edit pointer to the end of the line and then move the edit pointer back four characters. Display the current line, starting at the edit pointer.

You Type: +0 <4 L
Screen Shows: +0 <4 L
YYY

Truncate the line at the current position of the edit pointer. This command removes the YYY extension.

You Type: U
Screen Shows: U
INSERT A LINE

Go to the top of the text and display the contents of the buffer.

You Type: L *
Screen Shows: ^L *

MY FIRST LINE XXX
MY SECOND LINE YYY
INSERT A LINE
MY THIRD LINE YYY
FIFTH LINE
LAST LINE

Delete the current line and the next line.

You Type: D2
Screen Shows: D2

Move the edit pointer forward two lines.

You Type: +2
Screen Shows: +2
 INSERT A LINE

Delete this line.

You Type: D
Screen Shows: D
 INSERT A LINE

Display the current line.

You Type: L
Screen Shows: L
 MY THIRD LINE YYY

Move the edit pointer to the right three characters and display the text.

You Type: >3 L
Screen Shows: >3 L
 THIRD LINE YYY

Kill (delete) the 11 characters that constitute THIRD LINE.

You Type: K11
Screen Shows: K11
 THIRD LINE

Go to the beginning of the line and display it.

You Type: -0
Screen Shows: -0
 MY YYY

Concatenate (combine) two lines. Move the edit pointer to the end of the line; delete the character at the end of the line; move the edit pointer back to the beginning of the lines. Display the line.

You Type: +0 K -0
Screen Shows: 0 K -0
 MY YYYFIFTH LINE

Separate the two lines by inserting an end-of-line character.

You Type: >6 I / /
Screen Shows: >6 I / /
 MY YYY

Note: The end of line character is inserted before the current position of the edit pointer.

You Type: L
Screen Shows: L
FIFTH LINE

Sample Session 2

Clear the buffer by deleting its contents.

You Type: D*

Insert lines.

You Type:

Screen Shows: ONE TWO THREE 3.0
ONE TWO THREE 1.0
ONE
TWO
THREE
ONE TWO THREE 2.0
ONE
TWO
THREE
ONE TWO THREE 3.0

Go to the top of the text, and display all lines in the buffer.

You Type: L*

Screen Shows: ^L*
ONE TWO THREE 1.0
ONE
TWO
THREE
ONE TWO THREE 2.0
ONE
TWO
THREE
ONE TWO THREE 3.0

Search for the next occurrence of TWO.

You Type: S "TWO"
Screen Shows: S"TWO"
 ONE TWO THREE 1.0

Search for all occurrences of TWO that are between the edit pointer and the end of the buffer.

You Type: S*/TWO/
Screen Shows: S*/TWO/
 ONE TWO THREE 1.0
 TWO
 ONE TWO THREE 2.0
 TWO
 ONE TWO THREE 3.0

Go to the top of the buffer, and change the first occurrence of THREE to ONE.

You Type: C/THREE/ONE/
Screen Shows: ^ C/THREE/ONE/
 ONE TWO ONE 1.0

Move the edit pointer to the top of the buffer. Set the anchor to Column 2, and then use the search command to find each occurrence of TWO that begins in Column 2. Skip all other occurrences.

You Type: A2 S*/TWO/
Screen Shows: ^ A2 S*/TWO/
 TWO
 TWO

Move the edit pointer to the top of the buffer. Set the anchor to Column 1, and change each occurrence of ONE that begins in that column to XXX.

Note: ONE in Line 1 is not changed, since it does not begin in Column 1.

You Type: AC*/ONE/XXX/
Screen Shows: ^ AC*/ONE/XXX/
 XXX TWO ONE 1.0
 XXX
 XXX TWO THREE 2.0
 XXX
 XXX TWO THREE 3.0

Go to the top of the buffer, and display the text.

You Type: L *
Screen Shows: L *
XXX TWO ONE 1.0
XXX
TWO
THREE
XXX TWO THREE 2.0
XXX
TWO
THREE
XXX TWO THREE 3.0

Change the remaining ONE to XXX.

Note: The anchor is no longer set. It is reset to zero after each command is executed.

You Type: C / ONE / XXX /
Screen Shows: C / ONE / XXX /
XXX TWO XXX 1.0

Move to the beginning of the current line.

You Type: - 0
Screen Shows: - 0
XXX TWO XXX 1.0

Change three occurrences of XXX to ZZZ.

You Type: C3 / XXX / ZZZ /
Screen Shows: C3 / XXX / ZZZ /
ZZZ TWO XXX 1.0
ZZZ TWO ZZZ 1.0
ZZZ

Sample Session 3

Clear the buffer by deleting its contents:

You Type: D *

Display the directory of buffers and macros. The dollar sign (\$) identifies the secondary buffer as Buffer 0. The asterisk (*) identifies the primary buffer as Buffer 1. Edit has no macros defined. This is the initial environment when you start Edit.

You Type: .DIR
Screen Shows: .DIR

BUFFERS:
\$ 0
* 1

MACROS:

Insert some lines into Buffer 1 so that later you can identify it.

You Type:

Screen Shows: BUFFER ONE 1.0
BUFFER ONE 2.0
BUFFER ONE 3.0
BUFFER ONE 4.0

Display the text in Buffer 1.

You Type:
Screen Shows: ^L*
BUFFER ONE 1.0
BUFFER ONE 2.0
BUFFER ONE 3.0
BUFFER ONE 4.0

Make Buffer 0 the primary buffer. Buffer 1 becomes the secondary buffer.

You Type: B0
Screen Shows: B0

Display the directory of buffers and macros.

Note: The symbols identifying the buffers are now reversed.

You Type: .DIR
Screen Shows: .DIR

BUFFERS:
\$ 1
* 0

MACROS:

Insert some lines into Buffer 0.

You Type: BUFFER ZERO 1.0
 BUFFER ZERO 2.0
 BUFFER ZERO 3.0
 BUFFER ZERO 4.0
Screen Shows: BUFFER ZERO 1.0
 BUFFER ZERO 2.0
 BUFFER ZERO 3.0
 BUFFER ZERO 4.0

Display the text in Buffer 0.

You Type: L *
Screen Shows: ^L *
 BUFFER ZERO 1.0
 BUFFER ZERO 2.0
 BUFFER ZERO 3.0
 BUFFER ZERO 4.0

Switch to Buffer 1.

You Type: B
Screen Shows: B

Display the text in Buffer 1.

You Type: L *
Screen Shows: ^L *
 BUFFER ONE 1.0
 BUFFER ONE 2.0
 BUFFER ONE 3.0
 BUFFER ONE 4.0

Move the edit pointer to Line 3 in this buffer.

You Type: +2
Screen Shows: +2
 BUFFER ONE 3.0

Switch to Buffer 0.

You Type: B0
Screen Shows: B0

Display the text in Buffer 0.

You Type: L *
Screen Shows: L *
 BUFFER ZERO 1.0
 BUFFER ZERO 2.0
 BUFFER ZERO 3.0
 BUFFER ZERO 4.0

Move the edit pointer to Line 2 in this buffer.

You Type: +
Screen Shows: +
 BUFFER ZERO 2.0

Switch to Buffer 1.

You Type: B
Screen Shows: B

Display the text in Buffer 1 from the current position of the edit pointer.

Note: The position of the edit pointer has not changed since you switched to Buffer 0.

You Type: L *
Screen Shows: L *
 BUFFER ONE 3.0
 BUFFER ONE 4.0

Switch to Buffer 0.

You Type: B0
Screen Shows: B0

Display the text in Buffer 0 from the current position of the edit pointer.

Note: The position of the edit pointer has not changed since you switched to Buffer 1.

You Type: L *
Screen Shows: L *
 BUFFER ZERO 2.0
 BUFFER ZERO 3.0
 BUFFER ZERO 4.0

Delete the contents of Buffer 0.

You Type: D *
Screen Shows: ^D *
 BUFFER ZERO 1.0
 BUFFER ZERO 2.0
 BUFFER ZERO 3.0
 BUFFER ZERO 4.0

Make Buffer 1 the primary buffer and Buffer 0 the secondary buffer.

You Type: B
Screen Shows: B

Move two lines from the primary buffer (Buffer 1) into the secondary buffer (Buffer 0).

You Type: CTRL 7 P2 ENTER
Screen Shows: ^P2
BUFFER ONE 1.0
BUFFER ONE 2.0

Switch to Buffer 0, and show that the lines were moved to it.

You Type: B0 CTRL 7 L * ENTER
Screen Shows: B0 ^L *
BUFFER ONE 1.0
BUFFER ONE 2.0

Switch to Buffer 1. Go to the bottom of the buffer, and get the text out of the secondary buffer.

You Type: B/G * ENTER
Screen Shows: B/G *
BUFFER ONE 1.0
BUFFER ONE 2.0

Show the contents of the buffer.

Note: The order of the lines is changed as a result of moving the text.

You Type: CTRL 7 L * ENTER
Screen Shows: ^L *
BUFFER ONE 3.0
BUFFER ONE 4.0
BUFFER ONE 1.0
BUFFER ONE 2.0

Move two lines into the secondary buffer.

You Type: P2 ENTER
Screen Shows: P2
BUFFER ONE 3.0
BUFFER ONE 4.0

Move to the bottom of the buffer, and get the lines back out of the secondary buffer.

You Type: /G * ENTER
Screen Shows: /G *
BUFFER ONE 3.0
BUFFER ONE 4.0

Show that the order of the lines is restored.

You Type: CTRL 7 L *
Screen Shows: L *
BUFFER ONE 1.0
BUFFER ONE 2.0
BUFFER ONE 3.0
BUFFER ONE 4.0

Sample Session 4

Clear the buffer by deleting its contents:

You Type: CTRL 7 D * ENTER

Enter some lines of text.

You Type: LINE ONE ENTER
SECOND LINE OF TEXT ENTER
THIRD LINE OF TEXT ENTER
FOURTH LINE ENTER
FIFTH LINE ENTER
LAST LINE ENTER

Screen Shows: LINE ONE
SECOND LINE OF TEXT
THIRD LINE OF TEXT
FOURTH LINE
FIFTH LINE
LAST LINE

Open the file Oldfile for writing.

You Type: .WRITE"oldfile" ENTER
Screen Shows: .WRITE"oldfile"

Write all lines to the file.

You Type: CTRL 7 W * ENTER
Screen Shows: ^W *
LINE ONE
SECOND LINE OF TEXT
THIRD LINE OF TEXT
FOURTH LINE
FIFTH LINE
LAST LINE

END OF TEXT

Close the file.

You Type: .WRITE// ENTER
Screen Shows: .WRITE//

Verify that the buffer is empty.

You Type: `CTRL 7 L *` `ENTER`
Screen Shows: `^L *`

Open the file Oldfile for reading.

You Type: `.READ"oldfile"` `ENTER`
Screen Shows: `.READ"oldfile"`

Create a new file called Newfile for writing.

You Type: `.WRITE"newfile"` `ENTER`
Screen Shows: `.WRITE"newfile"`

Read four lines from the input file. The screen shows the lines as they are read in.

You Type: `R4` `ENTER`
Screen Shows: `R4`
`LINE ONE`
`SECOND LINE OF TEXT`
`THIRD LINE OF TEXT`
`FOURTH LINE`

Read all the remaining text from the file. The screen shows the lines. When there is no more text, the screen shows the *END OF FILE* message.

You Type: `R *` `ENTER`
Screen Shows: `R *`
`FIFTH LINE`
`LAST LINE`

`*END OF FILE*`

Go to the top of the buffer, and display the text to make sure it is inserted into the buffer.

You Type: `CTRL 7 L *` `ENTER`
Screen Shows: `^L *`
`LINE ONE`
`SECOND LINE OF TEXT`
`THIRD LINE OF TEXT`
`FOURTH LINE`
`FIFTH LINE`
`LAST LINE`

Write three lines to the output file, and display the lines.

You Type: w3
Screen Shows: w3
LINE ONE
SECOND LINE OF TEXT
THIRD LINE OF TEXT

Move to the next line and display it.

You Type: +
Screen Shows: +
FIFTH LINE

Show that when writing lines, the editor starts at the current line and not at the top of the buffer.

You Type: w
Screen Shows: w
FIFTH LINE

Go to the top of the buffer, and display the text to be sure that the lines were written to the output file.

You Type: L *
Screen Shows: ^L *
FOURTH LINE
LAST LINE

Clear the buffer.

You Type: D *
Screen Shows: ^D *
FOURTH LINE
LAST LINE

Switch to Buffer 2. Open the input file Oldfile, and read two lines from it.

You Type: B2 .READ"oldfile" R2
Screen Shows: B2 .READ"oldfile" R2
LINE ONE
SECOND LINE OF TEXT

Switch to Buffer 1. Open the input file Oldfile and read one line of text.

You Type: B .READ"oldfile" R
Screen Shows: B .READ"oldfile" R
LINE ONE

Switch to Buffer 2, and read one line.

Note: Your place in the file was not lost.

You Type: B2 R
Screen Shows: B2 R
THIRD LINE OF TEXT

Switch to Buffer 1, and read one line of text.

Note: Your place in the file was not lost.

You Type: B R
Screen Shows: B R
SECOND LINE OF TEXT

Switch to Buffer 2, and delete its contents.

You Type: B2 D*
Screen Shows: B2 ^D*
LINE ONE
SECOND LINE OF TEXT
THIRD LINE OF TEXT

Insert some extra lines into the buffer.

You Type: EXTRA LINE ONE
 EXTRA LINE TWO
Screen Shows: EXTRA LINE ONE
EXTRA LINE TWO

Try to write B2 buffer to file. It fails because you have not opened a file in this buffer.

You Type: W*
Screen Shows: ^W*
FILE CLOSED

Close the file for Buffer 1, and return to Buffer 2.

You Type: B .WRITE// B2
Screen Shows: B .WRITE// B2

Open the old "write" file for reading, and then read it back in.

You Type: .READ"newfile" R*
Screen Shows: .READ"newfile" R*
LINE ONE
SECOND LINE OF TEXT
THIRD LINE OF TEXT
FIFTH LINE

END OF FILE

Display the contents of the buffer.

Note: It read the file into the beginning of the buffer, since that was the position of the edit pointer.

You Type: L *
Screen Shows: ^L *
LINE ONE
SECOND LINE OF TEXT
THIRD LINE OF TEXT
FIFTH LINE
EXTRA LINE ONE
EXTRA LINE TWO

Sample Session 5

Delete all text from the edit buffer.

You Type: D *

Insert three lines.

You Type: LINE ONE
 LINE TWO
 LINE THREE
Screen Shows: LINE ONE
LINE TWO
LINE THREE

Create a new macro using an empty string.

You Type: .MAC //
Screen Shows: M:

Display the contents of the macro mode, which is now open.

Note: The E prompt is now M.

You Type: L *
Screen Shows: ^L *

Define the macro.

You Type: FIND
 S" "TWO"
Screen Shows: FIND
S" "TWO"

Display the contents of the macro.

You Type: L *
Screen Shows: ^L *
FIND
S" "TWO"

Close the macro's definition.

You Type: Q
Screen Shows: E :

Display the directory of buffers and macros.

You Type: .DIR
Screen Shows: .DIR
BUFFERS:
\$ 0
* 1

MACROS:
FIND

Display the contents of the edit buffer.

You Type: L *
Screen Shows: ^ L *
LINE ONE
LINE TWO
LINE THREE

Use the FIND macro to find the string TWO.

You Type: .FIND
Screen Shows: .FIND
LINE TWO

Reopen the definition of the FIND macro.

You Type: .MAC/FIND/
Screen Shows: .MAC/FIND/
M:

Show that the macro is still intact.

You Type: L *
Screen Shows: ^ L *
FIND
S"TWO"

Add the numeric parameter and the string parameter to the macro's header.

You Type: C/FIND/FIND #N \$STR/
Screen Shows: C/FIND/FIND #N \$STR/
FIND #N \$STR

Move to the second line of the macro.

You Type: +
Screen Shows: +
S"TWO"

Give the macro's parameters to the S command. Now the FIND macro will perform the same function as the S command.

You Type: C/"TWO"/ #N \$STR/
Screen Shows: C/"TWO"/ #N \$STR
 S #N \$STR

Close the macro's definition.

You Type: Q
Screen Shows: E:

Display the contents of the edit buffer.

You Type: L *
Screen Shows: ^L *
 LINE ONE
 LINE TWO
 LINE THREE

Use the FIND macro to find the next two occurrences of LINE.

You Type: .FIND 2 /LINE/
Screen Shows: .FIND 2 /LINE/
 LINE ONE
 LINE TWO

Create a new macro.

You Type: .MAC//
Screen Shows: .MAC//
 M:

Define the macro FIND_LINE, which performs the same function as the S command except that it returns the edit pointer to the head of the line after finding the last occurrence of STR.

You Type: FIND_LINE #N \$STR
Screen Shows: FIND_LINE #N \$STR

You Type: S #N \$STR
Screen Shows: S #N \$STR

Turn off the verify mode.

You Type: V
Screen Shows: V

Move the edit pointer to the first character of the current line.

You Type: -
Screen Shows: -

Close the macro's definition.

You Type: Q
Screen Shows: Q

E :

Display the contents of the edit buffer.

You Type: L *
Screen Shows: ^L *
LINE ONE
LINE TWO
LINE THREE

Use the FIND_LINE macro to search for the string TWO.

You Type: .FIND_LINE/TWO/
Screen Shows: .FIND_LINE/TWO/
LINE TWO

Show that the FIND_LINE macro left the edit pointer at the head of the line.

You Type: L
Screen Shows: L
LINE TWO

Create a new macro.

You Type: .MAC//
Screen Shows: .MAC//
M:

Use the exclamation point (!) command to comment itself. Type the following:

```

[ ] CONVERT_TO_LINES #N [ENTER]
[ ] !This is a comment [ENTER]
[ ] ! [ENTER]
[ ] !This macro converts the next n [ENTER]
[ ] !space characters to new line [ENTER]
[ ] !characters. [ENTER]
[ ] V0 ! Turn verify mode off [ENTER]
[ ] ! to prevent intermediate results [ENTER]
[ ] ! from being displayed. [ENTER]
[ ] ! [ENTER]
[ ] [ ! Begin loop [ENTER]
[ ] .SEARCH/ / ! Search for the space character. [ENTER]
[ ] !// ! Insert empty line (new line character). [ENTER]
[ ] - ! Back up one line. [ENTER]
[ ] C/ // ! Delete the next space character. [ENTER]
[ ] L + ! Show line, move past it. [ENTER]
[ ] ] #N ! End of loop. Repeat #N times. [ENTER]

```

Close the macro's definition.

You Type: Q [ENTER]
Screen Shows: Q
 E:

Display the contents of the edit buffer.

You Type: [CTRL] [7] L * [ENTER]
Screen Shows: ^L *
 LINE ONE
 LINE TWO
 LINE THREE

Convert all space characters to new line characters.

Note: The loop stops when the C command in the macro cannot find a space to delete.

You Type: .CONVERT_TO_LINES * [ENTER]
Screen Shows: .CONVERT_TO_LINES *
 LINE
 LINE
 LINE

Display the contents of the edit buffer.

You Type: `CTRL` `7` `L` * `ENTER`

Screen Shows: `^L *`
`LINE`
`ONE`
`LINE`
`TWO`
`LINE`
`THREE`

Edit Quick Reference Summary

EDIT OS-9 loads the editor and starts it without creating any read or write files. Perform text-file operations by opening files after the editor is running.

EDIT *newfile* OS-9 loads the editor and starts it. If *newfile* does not exist, Edit creates it and makes it the initial write file. Although this command does not create an initial read file, you can open read files after starting Edit.

EDIT *oldfile* OS-9 loads the editor and starts it, making the initial read file *oldfile*. The editor creates a new file called SCRATCH as the initial write file. When the edit session is complete, Edit deletes *oldfile* and renames SCRATCH to *oldfile*.

EDIT *oldfile newfile* OS-9 loads the editor and starts it. The initial read file is *oldfile*. The editor creates a file called *newfile* as the initial write file.

Edit Commands

.MACRO Executes the macro specified by the name following the period (.).

! Places comments inside a macro, and ignores the remainder of the command line.

␣ Inserts a line before the current position of the edit pointer.

ENTER Moves the edit pointer to the next line, and displays it.

+ *n* Moves the edit pointer forward *n* lines and displays the line.

-*n* Moves the edit pointer backward *n* lines and displays the line.

+0 Moves the edit pointer to the last character of the line.

-0	Moves the edit pointer to the first character of the current line and displays it.
>n	Moves the edit pointer forward <i>n</i> characters.
<n	Moves the edit pointer backward <i>n</i> characters.
CTRL] or] for external terminals	Moves the edit pointer to the beginning of the text.
/	Moves the edit pointer to the end of the text.
[commands] n	Repeats the sequence of commands between the two brackets <i>n</i> times.
:	Skips to the end of the innermost loop or macro if the fail flag is not on.
An	Sets the SEARCH/CHANGE anchor to Column <i>n</i> , restricting searches and changes to those strings starting in Column <i>n</i> . This command remains in effect for the current command line.
A0	Returns the anchor to the normal mode of searching so that strings are found regardless of the column in which they start.
Bn	Makes buffer <i>n</i> the primary buffer.
Cn str1 str2	Changes the next <i>n</i> occurrences of <i>str1</i> to <i>str2</i> .
Dn	Deletes <i>n</i> lines.
En str	Extends (adds the string to the end of) the next <i>n</i> lines.
Gn	Gets <i>n</i> lines from the secondary buffer, starting from the top. Inserts the lines before the current position in the primary buffer.
In str	Inserts a line containing <i>n</i> copies of the string before the current position of the edit pointer.
Kn	Kills <i>n</i> characters starting at the current position of the edit pointer.
Ln	Lists (displays) the next <i>n</i> lines, starting at the current position of the edit pointer.

Mn	Changes workspace (memory) size to <i>n</i> bytes.
Pn	Puts (moves) <i>n</i> lines from the position of the edit pointer in the primary buffer to the position of the edit pointer in the secondary buffer.
Q	Quits editing (and terminates editor). If you specified a file(s) when you entered Edit, Buffer 1 is written to the output file. The remainder of the input file is copied to the output file. All files are closed.
Rn	Reads <i>n</i> lines from the buffer's input file.
Sn str	Searches for the next <i>n</i> occurrences of the string.
Tn	Tabs to Column <i>n</i> of the present line. If <i>n</i> is greater than the line length, Edit extends the line with space.
U	Unextends (truncates) a line at the current position of the edit pointer.
Vmode	Turns the verify mode on or off.
Wn	Writes <i>n</i> lines to the buffer's output file.
Xn	Displays <i>n</i> lines that precede the edit position. The current line is counted as the first line.

Pseudo Macros

.CHANGE <i>n</i> <i>str1 str2</i>	Changes <i>n</i> occurrences of <i>str1</i> to <i>str2</i> .
.DEL <i>str</i>	Deletes the macro specified by <i>str</i> .
.DIR	Displays the directory of buffers and macros.
.EOB	Tests for the end of the buffer.
.EOF	Tests for the end of the file.
.EOL	Tests for the end of the line.
.F	Exits the innermost loop or macro and sets the fail flag.
.LOAD <i>str</i>	Loads macros from the path specified in the string.

.MAC <i>str</i>	Opens the macro specified by the string for definition. If you give an empty string, Edit creates a new macro.
.NEOB	Tests for not end of buffer.
.NEOF	Tests for not end of file.
.NEOL	Tests for not end of line.
.NEW	Writes all lines up to the current line to the initial output file, and then attempts to read an equal amount of text from the initial input file. The text read-in is appended to the end of the edit buffer.
.NSTR <i>str</i>	Tests to see if <i>string</i> does not match the characters at the current position of the edit pointer.
.READ <i>str</i>	Opens an OS-9 text file for reading, using <i>string</i> as the pathlist.
.S	Exits the innermost loop or macro and succeeds (clears the fail flag).
.SEARCH <i>n</i> <i>str</i>	Searches for <i>n</i> occurrences of <i>str</i> .
.SAVE <i>str1</i> <i>str2</i>	Saves the macros specified in <i>str1</i> on the file specified by the pathlist in <i>str2</i> .
.SHELL <i>command line</i>	Calls OS-9 shell to execute the command line.
.SIZE	Displays the size of memory used and the amount of memory available in the workspace.
.STAR <i>n</i>	Tests to see if <i>n</i> equals asterisk (infinity).
.STR <i>str</i>	Tests to see if <i>string</i> matches the characters at the current position of the edit pointer.
.WRITE <i>str</i>	Opens an OS-9 text for writing, using <i>str</i> as a pathlist.
.ZERO <i>n</i>	Tests <i>n</i> to see if it is zero.
[Starts at a macro loop; press CTRL 8 .
]	Ends at a macro loop; press CTRL 9 .

[^] Moves edit pointer to beginning of buffer; press **CTRL** **7**.

Editor Error Messages

BAD MACRO NAME You did not begin the first line in a macro with a legal name. You can close the definition of a macro after you give it a legal name.

BAD NUMBER You have entered an illegal numeric parameter, probably a number greater than 65,535.

BAD VAR NAME You have specified an illegal variable name, omitted the variable name, or included a \$ or # character in the commands parameter list.

BRACKET MISMATCH You have not entered brackets in pairs or the brackets are nested too deeply.

BREAK You pressed **CTRL** **C** or **E** to interrupt the editor. After printing the error message, the editor returns to command entry mode.

DUPL MACRO You attempted to close a macro definition with an existing macro name. Rename the macro before trying to close its definition.

END OF FILE You are at the end of the edit buffer.

FILE CLOSED You tried to write to a file that is not open. Either specify a write file when starting the editor from OS-9, or open an output file using the **.WRITE** pseudo macro.

MACRO IS OPEN You must close the macro definition before using the command.

MISSING DELIM The editor could not find a matching delimiter to complete the string you specified. You must put the entire string on one line.

NOT FOUND The editor cannot find the specified string or macro.

- UNDEFINED
VAR** You used a variable that is not specified in the macro's definition parameter list. A variable parameter can be used only in the macro in which it is declared.
- WHAT ??** The editor does not recognize a command. You typed a command that does not exist or misspelled a name.
- WORKSPACE
FULL** The buffer did not have room for the text you want to insert. Increase the workspace, or remove some text.

OS-9 Error Codes

The following table shows OS-9 error codes in hexadecimal and decimal. Error codes other than those listed are generated by programming languages or user programs.

OS-9 Error Codes

Code		Code Meaning
HEX	DEC	
\$01	001	UNCONDITIONAL ABORT. An error occurred from which OS-9 cannot recover. All processes are terminated.
\$02	002	KEYBOARD ABORT. You pressed BREAK to terminate the current operation.
\$03	003	KEYBOARD INTERRUPT. You pressed SHIFT BREAK either to cause the current operation to function as a background task with no video display or to cause the current task to terminate.
\$B7	183	ILLEGAL WINDOW TYPE. You tried to define a text type window for graphics or used illegal parameters.
\$B8	184	WINDOW ALREADY DEFINED. You tried to create a window that is already established.
\$B9	185	FONT NOT FOUND. You tried to use a window font that does not exist.
\$BA	186	STACK OVERFLOW. Your process (or processes) requires more stack space than is available on the system.
\$BB	187	ILLEGAL ARGUMENT. You have used an argument with a command that is inappropriate.
\$BD	189	ILLEGAL COORDINATES. You have given coordinates to a graphics command which are outside the screen boundaries.

Code		Code Meaning
HEX	DEC	
\$BE	190	INTERNAL INTEGRITY CHECK. System modules or data are changed and no longer reliable.
\$BF	191	BUFFER SIZE IS TOO SMALL. The data you assigned to a buffer is larger than the buffer.
\$C0	192	ILLEGAL COMMAND. You have issued a command in a form unacceptable to OS-9.
\$C1	193	SCREEN OR WINDOW TABLE IS FULL. You do not have enough room in the system window table to keep track of any more windows or screens.
\$C2	194	BAD/UNDEFINED BUFFER NUMBER. You have specified an illegal or undefined buffer number.
\$C3	195	ILLEGAL WINDOW DEFINITION. You have tried to give a window illegal parameters.
\$C4	196	WINDOW UNDEFINED. You have tried to access a window that you have not yet defined.
\$C8	200	PATH TABLE FULL. OS-9 cannot open the file because the system path table is full.
\$C9	201	ILLEGAL PATH NUMBER. The path number is too large, or you specified a non-existent path.
\$CA	202	INTERRUPT POLLING TABLE FULL. Your system cannot handle an interrupt request, because the polling table does not have room for more entries.
\$CB	203	ILLEGAL MODE. The specified device cannot perform the indicated input or output function.
\$CC	204	DEVICE TABLE FULL. The device table does not have enough room for another device.

Code		Code Meaning
HEX	DEC	
\$CD	205	ILLEGAL MODULE HEADER. OS-9 cannot load the specified module because its sync code, header parity, or cyclic redundancy code is incorrect.
\$CE	206	MODULE DIRECTORY FULL. The module directory does not have enough room for another module entry.
\$CF	207	MEMORY FULL. Process address space is full or your computer does not have sufficient memory to perform the specified task.
\$D0	208	ILLEGAL SERVICE REQUEST. The current program has issued a system call containing an illegal code number.
\$D1	209	MODULE BUSY. Another process is already using a non-shareable module.
\$D2	210	BOUNDARY ERROR. OS-9 has received a memory allocation or deallocation request that is not on a page boundary.
\$D3	211	END OF FILE. A read operation has encountered an end-of-file character and has terminated.
\$D4	212	RETURNING NON-ALLOCATED MEMORY. The current operation has attempted to deallocate memory not previously assigned.
\$D5	213	NON-EXISTING SEGMENT. The file structure of the specified device is damaged.
\$D6	214	NO PERMISSION. The attributes of the specified file or device do not permit the requested access.
\$D7	215	BAD PATH NAME. The specified pathlist contains a syntax error, for instance an illegal character.
\$D8	216	PATH NAME NOT FOUND. The system cannot find the specified pathlist.

Code		Code Meaning
HEX	DEC	
\$D9	217	SEGMENT LIST FULL. The specified file is too fragmented for further expansion.
\$DA	218	FILE ALREADY EXISTS. The specified file-name already exists in the specified directory.
\$DB	219	ILLEGAL BLOCK ADDRESS. The file structure of the specified device is damaged.
\$DC	220	PHONE HANGUP - DATA CARRIER DETECT LOST. The data carrier detect is lost on the RS-232 port.
\$DD	221	MODULE NOT FOUND. The system received a request to link a module that is not in the specified directory.
\$DF	223	SUICIDE ATTEMPT. The current operation has attempted to return to the memory location of the stack.
\$E0	224	ILLEGAL PROCESS NUMBER. The specified process does not exist.
\$E2	226	NO CHILDREN. The system has issued a <i>wait service</i> request but the current process has no dependent process to execute.
\$E3	227	ILLEGAL SWI CODE. The system received a software interrupt code that is less than 1 or greater than 3.
\$E4	228	PROCESS ABORTED. The system received a signal Code 2 to terminate the current process.
\$E5	229	PROCESS TABLE FULL. A fork request cannot execute because the process table has no room for more entries.
\$E6	230	ILLEGAL PARAMETER AREA. A fork call has passed incorrect high and low bounds.
\$E7	231	KNOWN MODULE. The specified module is for internal use only.

Code		Code Meaning
HEX	DEC	
\$E8	232	INCORRECT MODULE CRC. The cyclic redundancy code for the module being accessed is bad.
\$E9	233	SIGNAL ERROR. The receiving process has a previous, unprocessed signal pending.
\$EA	234	NON-EXISTENT MODULE. The system cannot locate the specified module.
\$EB	235	BAD NAME. The specified device, file, or module name is illegal.
\$EC	236	BAD MODULE HEADER. The specified module header parity is incorrect.
\$ED	237	RAM FULL. No free system random access memory is available: the system address space is full, or there is no physical memory available when requested by the operating system in the system state.
\$EE	238	UNKNOWN PROCESS ID. The specified process ID number is incorrect.
\$EF	239	NO TASK NUMBER AVAILABLE. All available task numbers are in use.

Device Driver Errors

I/O device drivers generate the following error codes. In most cases, the codes are hardware-dependent. Consult your device manual for more details.

Code		Code Meaning
HEX	DEC	
\$F0	240	UNIT ERROR. The specified device unit doesn't exist.
\$F1	241	SECTOR ERROR. The specified sector number is out of range.
\$F2	242	WRITE PROTECT. The specified device is write-protected.

Code		Code Meaning
HEX	DEC	
\$F3	243	CRC ERROR. A cyclic redundancy code error occurred on a read or write verify.
\$F4	244	READ ERROR. A data transfer error occurred during a disk read operation, or there is a SCF (terminal) input buffer overrun.
\$F5	245	WRITE ERROR. An error occurred during a write operation.
\$F6	246	NOT READY. The device specified has a <i>not ready</i> status.
\$F7	247	SEEK ERROR. The system attempted a seek operation on a non-existent sector.
\$F8	248	MEDIA FULL. The specified media has insufficient free space for the operation.
\$F9	249	WRONG TYPE. An attempt is made to read incompatible media (for instance an attempt to read double-side disk on single-side drive).
\$FA	250	DEVICE BUSY. A non-shareable device is in use.
\$FB	251	DISK ID CHANGE. You changed diskettes when one or more files are open.
\$FC	252	RECORD IS LOCKED-OUT. Another process is accessing the requested record.
\$FD	253	NON-SHARABLE FILE BUSY. Another process is accessing the requested file.

Color Computer 2 Compatibility

Color Computer 3 OS-9 Level Two provides compatibility with the Color Computer 2 and OS-9 Level One by letting you use the video display in the Alphanumeric mode (including *Semigraphic* box graphics) and in the Graphics mode. To control the display, it has many built-in functions that you activate using ASCII control characters. Any program written in a language using standard output statements (such as PUT in BASIC) can use these functions. Color Computer BASIC09 has a Graphics Interface Module that can automatically generate most of these codes using BASIC09 RUN statements.

The Color Computer's display system uses a separate memory area for each Display mode. Therefore, operations on the Alpha display do not affect the Graphics display and vice-versa. You can select either display with software control. (See *Getting Started With Extended Color BASIC* for more detailed information.)

The system interprets 8-bit characters sent to the display according to their numerical values, as shown in this chart:

Character Range (Hex)	Mode/Function
--------------------------------------	----------------------

00 - 0E	Alpha—Cursor and screen control.
0F - 1D	Graphics—Drawing and screen control.
1B	Alpha, Graphics—Changing Palette colors.

Alpha mode:

1B 31 2 h change cursor color

1B 31 c h change foreground color

1B 31 d h change background color

where h is a hex number from 0 to 3F (0 to 63 decimal) which determines the color.

Character Range (Hex)	Mode/Function
--------------------------------------	----------------------

Graphics mode:

1B 31 pr h changes foreground/
 background color

where pr is a palette register # (0 - F,
hex)

where h is a hex number from 0 to 3F (0
to 63 decimal) which determines the
color.

- | | |
|---------|-----------------------------|
| 20-5F | Alpha—Uppercase characters. |
| 60 - 7F | Alpha—Lowercase characters. |
| 80 - FF | Alpha—Semigraphic patterns. |

The device driver CC3IO calls a subroutine module named VDGInt to handle all text and graphics for the Color Computer 2 compatibility mode.

Alpha Mode Display

The Alpha mode is the standard operational mode. Use it to display alphanumeric characters and semigraphic box graphics. Use it also to simulate the operation of a typical computer terminal with functions for scrolling, cursor positioning, clearing the screen, deleting lines, and so on.

The Alpha mode assumes that each 8-bit code the system sends to the display is an ASCII character. If the high-order bit of the code is clear, the system displays the appropriate alphanumeric character. If the high-order bit is set, OS-9 generates a Semigraphic 6 graphics box. See *Getting Started With Extended Color BASIC* for an explanation of semigraphic functions.

The standard 32-column Alpha mode display is handled by the I/O subroutine module VDGInt. CC3IO calls this module (included in the standard boot file) to process all text and semigraphic output.

The following chart provides codes for screen display and cursor control. You can use the functions from the OS-9 system prompt by typing DISPLAY, followed by the appropriate codes. For instance, to clear the screen, type:

```
display 0c 
```

To position the cursor at column 16, Line 5 and display the word HELLO, type:

```
display 02 30 25 48 45 4c 4c 4f 
```

You can also use the following codes in a language, such as BASIC09. To do so, use decimal numbers with the CHR\$ function, such as:

```
print chr$(02);chr$(48);chr$(37);chr$(72)  
;chr$(69);chr$(76);chr$(76);chr$(79)
```

Using Alpha Mode Controls with Windows

The control functions in the following chart also function properly under the high resolution windowing systems. References to "screen" are also references to windows.

Alpha Mode Command Codes

Hex Control Code	Decimal Control Code	Name/Function
\$01	01	HOME—Returns the cursor to the upper left corner of the screen.
\$02	02	CURSOR XY—Moves the cursor to character X of line Y. To arrive at the values for X and Y, add 20 hexadecimal to the location where you want to place the cursor. For example, to position the cursor at Character 5 of Line 10 (hexadecimal A), do these calculations: $\begin{array}{r} 5 \\ + 20 \\ = 25 \text{ hexadecimal} \end{array}$ $\begin{array}{r} 0A \\ + 20 \\ = 2A \text{ hexadecimal} \end{array}$ The two coordinates are \$25 and \$2A.
\$03	03	ERASE LINE—Erases all characters on the line occupied by the cursor.
\$04	04	CLEAR TO END OF LINE—Erases all characters from the cursor position to the end of the line.

Hex Control Code	Decimal Control Code	Name/Function
\$05	05	CURSOR ON-OFF—Allows alteration of the cursor based on the value of the next character. Codes are as follow:
		Default Color
	Hex	Dec
	Char	Function
	\$20	32
	space	Cursor OFF
	\$21	33
	!	Cursor ON
	\$22	34
	"	Cursor ON
	\$23	35
	#	Cursor ON
	\$24	36
	\$	Cursor ON
	\$25	37
	%	Cursor ON
	\$26	38
	&	Cursor ON
	\$27	39
	'	Cursor ON
	\$28	40
	(Cursor ON
	\$29	41
)	Cursor ON
	\$2A	42
	*	Cursor ON
\$06	06	CURSOR RIGHT—Moves the cursor to the right one character position.
\$07	07	BELL—Sounds a bell (beep) through monitor speaker.
\$08	08	CURSOR LEFT—Moves the cursor to the left one character position.
\$09	09	CURSOR UP—Moves the cursor up one line.
\$0A	10	CURSOR DOWN (linefeed)—Moves the cursor down one line.
\$0C	12	CLEAR SCREEN—Erases the entire screen, and homes the cursor (positions it at the upper left corner of the screen).
\$0D	13	RETURN—Returns the cursor to the leftmost character on the line.
\$0E	14	DISPLAY ALPHA—Switches the screen from Graphic mode to Alphanumeric mode.

Graphics Mode Display

Use the Graphics mode to display high-resolution 2- or 4-color VDG graphics. The Graphics mode includes commands to set color, plot and erase individual points, draw and erase lines, position the graphics cursor, and draw circles.

You must execute the *display graphics* command before using any other Graphics mode command. This command displays the graphics screen and sets a display format and color.

The first time you enter the display graphics command, OS-9 allocates a 6144-byte display memory. There must be at least that much contiguous free memory available. (You can use MFREE to check free memory.) The system retains the display memory until you give the *end graphics* command, even if the program that initiated the Graphics mode finishes. Always use the end graphics command to release the display memory when you no longer need the Graphics mode.

Graphics mode supports two basic formats. The 2-color format has 256 horizontal by 192 vertical points (G6R mode). The 4-color format has 128 horizontal by 192 vertical points (G6C mode). Either mode provides both color sets. Regardless of the resolution of the selected format, all Graphics mode commands use a 256 by 192 point coordinate system. The X and Y coordinates are always positive numbers. Point 0,0 is the lower left corner of screen.

Many commands use an invisible *graphics cursor* to reduce the output required to generate graphics. You can explicitly set this cursor to any point by using the *set graphics cursor* command. You can also use any other commands that include x,y coordinates (such as *set point*) to move the graphics cursor to the specified position.

Any graphics function that *draws* on the graphics screen requires that the VDGInt module is loaded into memory during the system boot.

Graphics Mode Selection Codes

Code	Format
00	256 x 192 two-color graphics
01	128 x 192 four-color graphics

Color Set and Foreground Color Selection Codes

		2-Color Format		4-Color Format	
	Char	Back-ground	Fore-ground	Back-ground	Fore-ground
Color Set 0	00	Black	Black	Green	Green
	01	Black	Green	Green	Yellow
	02			Green	Blue
	03			Green	Red
Color Set 1	04	Black	Black	Buff	Buff
	05	Black	Buff	Buff	Cyan
	06			Buff	Magenta
	07			Buff	Orange
Color Set 2	08			Black	Black
	09			Black	Dark Green
	10			Black	Med. Green
	11			Black	Light Green
Color Set 3	12			Black	Black
	13			Black	Green
	14			Black	Red
	15			Black	Buff

Graphics Mode Control Commands

Hex Control Code	Decimal Control Code	Name/Function
\$0F	15	DISPLAY GRAPHICS—Switches the screen to the Graphics mode. Use this command before any other graphics commands. The first time you use it, the system assigns a 6-kilobyte display buffer for graphics. If 6K of contiguous memory isn't available, OS-9 displays an error. Follow the <i>display graphics</i> command with two characters specifying the Graphics mode and color/color set, respectively.
\$10	16	PRESET SCREEN—Presets the entire screen to the color code passed by the next character.

Hex Control Code	Decimal Control Code	Name/Function
\$11	17	SET COLOR—Sets the foreground color (and color set) to the color specified by the next character but does not change the Graphics mode.
\$12	18	END GRAPHICS—Disables the Graphics mode, returns the 6K byte graphics memory area to OS-9 for other use, and switches to Alpha mode.
\$13	19	ERASE GRAPHICS—Erases all points by setting them to the background color, and positions the graphics cursor at the desired position.
\$14	20	HOME GRAPHICS CURSOR—Moves the graphics cursor to coordinates 0,0 (the lower left corner).
\$15	21	SET GRAPHICS CURSOR—Moves the graphics cursor to the given x,y coordinates. For x and y, the system uses the binary value of the two characters that immediately follow.
\$16	22	DRAW LINE—Draws a line in the foreground color from the graphics cursor position to the given x,y coordinates. For x and y, the system uses the binary value of the two characters that immediately follow. The graphics cursor moves to the end of the line.
\$17	23	ERASE LINE—Operates the same as the <i>draw line</i> function, except that OS-9 draws the line in the background color, thus erasing the line.
\$18	24	SET POINT—Sets the pixel at point x,y to the foreground color. For x and y, the system uses the binary values of the two characters that immediately follow. The graphics cursor moves to the point set.

Hex Control Code	Decimal Control Code	Name/Function
\$19	25	ERASE POINT—Operates the same as the <i>set point</i> function, except that OS-9 draws the point in the background color, thus erasing the point.
\$1A	26	DRAW CIRCLE—Draws a circle in the foreground color using the graphics cursor as the center point and using the the binary value of the next character as the radius.
\$1C	28	ERASE CIRCLE—Operates the same as the <i>draw circle</i> function, except that OS-9 draws the circle in the background color, thus erasing the circle.
\$1D	29	FLOOD FILL— <i>paints</i> with the foreground color, starting at the graphics cursor position and extending over adjacent pixels having the same color as the pixel under the graphics cursor.

Note: When you call FILL the first time, it requests allocation of a 512-byte stack for the fill routine. The system does not return this memory until you terminate graphics with the *end graphics* command.

Note: The chart uses hexadecimal codes for compatibility with the OS-9 DISPLAY command.

Display Control Codes Summary

1st Byte		2nd Byte	3rd Byte	Function
Dec	Hex			
00	00			Null
01	01			Home alpha cursor
02	02	Column + 32	Row + 32	Position alpha cursor
03	03			Erase line

OS-9 Commands Reference

1st Byte		2nd Byte	3rd Byte	Function
Dec	Hex			
04	04			Erase to End of line
05	05	Cursor Code		Alter Cursor
06	06			Move cursor right
07	07			Sound terminal bell
08	08			Move cursor left
09	09			Move cursor up
10	0A			Move cursor down
11	0B			Erase to End of Screen
12	0C			Clear screen
13	0D			Carriage return
14	0E			Select Alpha mode
15	0F	Mode	Color Code	Select Graphics mode
16	10	Color Code		Preset screen
17	11	Color Code		Select color
18	12			Quit Graphics mode
19	13			Erase screen
20	14			Home Graphics cursor
21	15	X Coord	Y Coord	Move graphics cursor
22	16	X Coord	Y Coord	Draw line to x/y
23	17	X Coord	Y Coord	Erase line to x/y
24	18	X Coord	Y Coord	Set point at x/y
25	19	X Coord	Y Coord	Clear point at x/y
26	1A	Radius		Draw circle
28	1C	Radius		Erase circle
29	1D			Flood Fill

OS-9 Keyboard Codes

Key Definitions With Hexadecimal Values

NORM	SHFT	CTRL	NORM	SHFT	CTRL	NORM	SHFT	CTRL
0	30	0	30	--	@	40	60	NUL 00
1	31	!	21		7C	a	61	A 41 SOH 01
2	32	"	22		00	b	62	B 42 STX 02
3	33	#	23	-	7E	c	63	C 43 ETX 03
4	34	\$	24		00	d	64	D 44 EOT 04
5	35	%	25		00	e	65	E 45 EMD05
6	36	&	26		00	f	66	F 46 ACK 06
7	37	'	27	^	5E	g	67	G 47 BEL 07
8	38	(28	[5B	h	68	H 48 BSP 08
9	39)	29]	5D	i	69	I 49 HT 09
:	3A	*	2A		00	j	6A	J 4A LF 0A
;	3B	+	2B		00	k	6B	K 4B VT 0B
,	2C	<	3C	{	7B	l	6C	L 4C FF 0C
-	2D	=	3D		5F	m	6D	M 4D DR 0D
.	2E	>	3E	}	7D	n	6E	N 4E CO 0E
/	2F	?	3F	\	5C	o	6F	O 4F CI 0F
						p	70	P 50 DLE 10
						q	71	Q 51 DC1 11
						r	72	R 52 DC2 12
						s	73	S 53 DC3 13
						t	74	T 54 DC4 14
						u	75	U 55 NAK 15
						v	76	V 56 SYN 16
						w	77	W 57 ETB 17
						x	78	X 58 CAN 18
						y	79	Y 59 EM 19
						z	7A	Z 5A SUM 1A

Function Keys

	NORM	SHFT	CTRL
BREAK	05	03	1B
ENTER	0D	0D	0D
SPACE	20	20	20
←	08	18	10
→	09	19	11
↓	0A	1A	12
↑	0C	1C	13

OS-9 Keyboard Control Functions

Key Definitions for Special Functions and Characters

Key Combination	Control Function or Character
ALT	Alternate key—Sets the high order bit on a character. Press ALT <i>char</i> .
CTRL	Use as a control key.
BREAK or CTRL E	Stops the program currently executing.
CTRL -	Generates an underscore (_).
CTRL ,	Generates a left brace ({}).
CTRL .	Generates a right brace ({}).
CTRL /	Generates a reverse slash (\).
CTRL BREAK	Generates an end-of-file (EOF). This sequence is the same as pressing ESC on a standard terminal.
← or CTRL H	Generates a backspace.
SHIFT ← or CTRL X	Deletes the entire current line.
SHIFT BREAK or CTRL C	Interrupts the video display of a running program. This sequence reactivates the shell and then runs the program as a background task.
CTRL 0	Upper-/lowercase shift lock function.
CTRL 1	Generates a vertical bar () in reverse video.
CTRL 3	Generates a tilde (~) character.
CTRL 7	Generates an up arrow or caret (^).
CTRL 8	Generates a left bracket ([).
CTRL 9	Generates a right bracket (]).

Key Combination	Control Function or Character
CTRL A	Repeats the previous command line.
CTRL D	Redisplays the command line.
CTRL W	Temporarily halts output to the screen. Press any key to resume output.
CTRL CLEAR	Enable/Disable Keyboard mouse.
CLEAR	Change screens.
SHIFT CLEAR	Change screens in reverse order.
ENTER	

Index

- ACIAPAK 5-6, 5-7, 6-96
- active state 4-2
- address 2-4
 - memory 4-5
- allocate memory for devices 6-55
- alpha mode B-3
 - select B-10
- alphanumeric mode B-1
- ampersand separator 3-6
- append files 6-68
- application program 1-3
- arglist 6-2
- ASCII 2-5
 - control characters B-1
 - convert 6-38
- ASM 3-2
- asterisk, editor 7-3
- ATTR 2-10, 6-5
- attribute 2-5, 2-8, 2-10, 6-5
- auto-answer modem 6-96, 6-108
- background
 - color B-7
 - process 3-7
 - task 4-1
 - screen 5-2
- backspace 6-93
 - character 6-94, 6-106, 6-107
 - editor 7-2
 - over line 6-93, 6-106
- BACKUP 5-4, 6-7
- backup a directory 6-39
- BASIC09 2-5, 2-6, 3-13, B-1
- baud rate 5-4, 5-5, 5-6, 6-96, 6-98, 6-109
- begin a window 6-103
- bell
 - character 6-95, 6-108
 - sound B-10
- bit 2-1
 - stop 5-5, 5-6
 - user 2-11
- bitmap 2-5
- block
 - number 4-5
 - devices 1-2
- bootstrap 5-1
 - file 5-2
- box graphics B-3
- brackets 6-3
- buffer 3-7, 7-2
 - edit 7-1
 - secondary 7-1
 - text 7-1
- BUILD 2-6, 3-10, 6-10, 6-71, 6-75
- built-in commands 3-1, 3-11
- byte 2-1
- carriage return B-10
- CC3Disk 5-1
- CC3Go 5-2
- CC3IO 5-1, B-2
- chaining programs 6-44
- change
 - attributes 2-10, 2-11
 - directory 6-12, 6-91, 6-84
 - file name 6-84
 - priority 3-12, 6-88
 - system parameters 6-93
- character
 - ASCII 2-5
 - delete 6-107
 - devices 1-2
 - backspace 6-94, 6-106, 6-107
 - bell 6-95, 6-108
 - delete line 6-94, 6-107
 - dup 6-95, 6-107
 - end-of-file 6-94, 6-107

- character (*cont'd*)
 - end-of-record 6-94, 6-107
 - lowercase B-1
 - pause 6-95, 6-108
 - quit 6-95, 6-108
 - reprint 6-95, 6-107
 - terminate 6-95, 6-108
 - uppercase B-2
- CHD 3-11, 6-12
- check disk structure 6-25
- child process 3-6, 4-2
- CHX 3-11, 6-12
- circle
 - draw B-9
 - erase B-9, B-10
- clear
 - screen B-5
 - to end-of-line B-4
- clock 5-2
- cluster 2-4, 2-5
- CMDS directory 5-1, 5-3, 5-4
- CMP 6-14
- COBBLER 6-16, 6-72
- code
 - alpha mode control B-4
 - cursor B-5
 - object 2-7
 - position-independent 4-8
 - re-entrant 4-6
- color
 - background B-7
 - foreground B-7, B-8
 - select B-10
 - set, graphics B-7
- combine files 6-68
- command
 - grouping 3-2, 3-9
 - help 6-51
 - interpreter 6-90
 - line 3-1, 3-2
 - parameters, editor 7-3
 - separator 3-1, 3-5
 - summary, editor 7-55
- command codes
 - alpha mode B-4
 - graphics B-7
- commandname 6-2
- commands
 - ASM 3-2
 - ATTR 2-10, 2-11, 6-5
 - BACKUP 6-7
 - BUILD 2-6, 3-10, 6-10, 6-71, 6-75
 - built-in 3-11
 - CHD 3-11, 6-12
 - CHX 3-11, 6-12
 - CMP 6-14
 - COBBLER 6-16, 6-72
 - CONFIG 5-2, 5-3, 5-4, 6-18
 - COPY 2-3, 3-6, 4-8, 6-22
 - DATE 6-24
 - DCHECK 6-25
 - DEINIZ 6-30
 - DEL 6-31
 - DELDIR 2-3, 6-33
 - DIR 2-6, 2-9, 6-35
 - DISPLAY 6-38
 - DSAVE 6-39
 - DUMP 2-8, 6-72
 - ECHO 6-42
 - edit macro 7-28
 - editor 7-2
 - ERROR 5-2, 6-43
 - EX 3-11, 6-44
 - FORMAT 6-46
 - FREE 6-49
 - GET 2-6
 - HELP 6-51
 - i 3-11
 - IDENT 3-3, 6-52
 - INIZ 6-55
 - KILL 3-12, 6-56
 - LINK 6-58
 - LIST 2-3, 2-5, 2-8, 3-4, 6-59
 - LOAD 4-7, 6-61

-
- commands (*cont'd*)
- MAKDIR 2-3, 2-11, 6-63
 - MDIR 6-64
 - MERGE 6-68
 - MFREE 6-69
 - MODPATCH 6-70
 - MONTYPE 6-74
 - OS9GEN 5-2, 5-3, 6-76
 - p 3-12
 - PROCS 3-7, 4-2, 6-80
 - PUT 2-6
 - PWD 6-82
 - PXD 6-82
 - RENAME 6-84
 - RUNB 3-13
 - SEEK 2-6
 - SETIME 5-3, 6-86
 - SETPR 3-12, 6-88
 - SHELL 3-6, 6-90
 - t 3-12
 - TMODE 6-93
 - TUNEPORT 6-98
 - UNLINK 4-7, 4-8, 6-100
 - w 3-12
 - WCREATE 6-103
 - x 3-12
 - XMODE 5-4, 5-5, 5-7, 6-106
- comment, in a program 3-12
- compare files 6-14
- concurrent
 - execution 3-5, 6-91
 - mode 3-10
 - process 3-9
 - task 4-1
- CONFIG 5-2, 5-3, 5-4, 6-18
- control
 - characters, ASCII B-1
 - keys, editor 7-2
- convert to ASCII 6-38
- COPY 2-3, 3-6, 4-8, 6-22
- copy
 - a directory 6-39
 - diskettes 6-7
- CPU 4-1
 - priority 6-88
- CRC 2-7, 6-71
- create
 - a directory 6-63
 - a file 6-10
 - OS9Boot 6-16, 6-76
 - process 3-6
 - system diskette 5-3, 5-4, 6-16, 6-18, 6-76
- current
 - directory 4-4, 6-12
 - processes 6-80
- cursor
 - on/off B-5
 - codes B-5
 - control B-1, B-4
 - graphics B-6, B-8
 - home B-4
 - move B-5, B-10
- cyclic redundancy checksum 2-7
- data format 2-1
- data output, halt 7-3
- data
 - redirect 3-4
 - input/output 1-2
 - passing 4-4
 - process 2-1
 - sending 2-1
 - transfer 2-1
- DATE 6-24
- date 2-5
 - set 6-86
- day 6-2
- DCHECK 6-25
- deallocate a device 6-30
- DEINIZ 6-30
- DEL 6-31
- delay, not ready 5-6
- DELDIR 2-3, 6-33
-

delete

- a character 6-107
- a directory 6-33
- a line 7-3
- a memory module 4-7, 6-100
- files 6-31
- line character 6-94, 6-107
- lines, editor 7-10

descriptor

- device 1-2
- file 2-3

detach a device 6-30

device

- allocate memory 6-55
- block-oriented 1-2
- character 1-2
- deallocate 6-30
- descriptor 1-2, 5-1
- driver 1-2, 2-1, 5-1
- driver initialization 6-55
- name 2-12, 2-13
- window 2-12 - 2-13

devname 6-2

DIR 2-6, 2-9, 6-35

directory 2-2, 2-3

- attribute 2-8
- change 6-12, 6-91
- change name 6-84
- CMDS 5-1, 5-3, 5-4
- copy 6-39
- create 6-63
- current 4-4, 6-12
- delete 6-33
- list 6-35
- module 4-6
- ownership 2-8
- SYS 5-1, 5-4
- view 6-82
- working 6-12

dirname 6-2

disable echo 6-94, 6-107

disk

- cluster 2-4
- file 2-3, 2-4
- I/O 3-8
- initialization 6-46
- names 2-12
- ownership 2-8
- sector 2-4
- structure, check 6-25
- raw I/O 3-8
- unused sectors 6-49

diskette

- copy 6-7
- density 2-5
- tracks 2-5
- system 2-2

DISPLAY 6-38

display

- a directory 6-35
- current processes 6-80
- date and time 6-24
- error message 6-43
- execution directory 6-82
- file contents 6-59
- free memory 6-69
- graphics B-7
- help 6-51
- memory module names 6-64
- messages 6-42
- on next line 7-2
- text, editor 7-6
- unused disk sectors 6-49
- working directory 6-82

double density 2-5

draw

- a circle B-9
- a line B-8, B-10

drivers, device 1-2

DSAVE 6-39

DUMP 2-8, 6-72

dup character 6-95, 6-107

duplicate

- last line 6-95
- line 6-107

-
- ECHO 6-42
 - echo 6-92
 - enable/disable 6-94, 6-107
 - edit
 - buffer 7-1
 - commands, EDIT 7-5
 - pointer 7-1, 7-2, 7-7
 - EDIT, editor 7-5
 - editor 7-1
 - backspace 7-2
 - command summary 7-55
 - command syntax 7-4
 - commands 7-2
 - control keys 7-2
 - delete lines 7-10
 - error messages 7-59
 - getting started 7-4
 - insert lines 7-10
 - interrupt 7-3
 - numeric parameters 7-3
 - quick reference 7-55
 - searching 7-13
 - substituting 7-13
 - terminate 7-2
 - text file operations 7-18
 - using the asterisk 7-3
 - ellipsis 6-3
 - enable echo 6-94, 6-101
 - end graphics B-8
 - end-of-file
 - terminate 7-2
 - character 6-94, 6-101
 - end-of-line
 - clearing B-4
 - erase B-10
 - end-of-record character 6-94, 6-107
 - erase
 - a circle B-9, B-10
 - a line B-10
 - graphics B-8
 - line B-4, B-8, B-9
 - point B-9
 - erase (*cont'd*)
 - to end-of-line B-10
 - Errmsg 5-2
 - ERROR 5-2, 6-43
 - error 3-12, 6-92
 - message file 5-2
 - messages, editor 7-59
 - output 6-91
 - path 3-4
 - establish a directory 6-63
 - EX 3-11, 6-44
 - exclamation mark separator 3-8
 - execute
 - a program 6-90
 - permission 2-9, 2-10
 - execution
 - concurrent 3-5, 6-91
 - modifier 3-1, 3-3
 - sequential 3-5, 3-6, 6-91
 - fields 2-6
 - file 2-2 - 2-4
 - attribute 2-8
 - change name 6-84
 - compare 6-14
 - copy 6-22
 - create 6-10
 - delete 6-31
 - descriptor 2-3
 - descriptor sector 2-5
 - display contents 6-59
 - load in memory 6-61
 - merge 6-68
 - manager 5-1
 - OS9Boot 5-4
 - ownership 2-8
 - pointer 2-4
 - procedure 2-6, 3-10, 3-11
 - random access 2-6
 - security 2-8
 - single-user 2-8
 - size 2-5
-

- file (*cont'd*)
 - Startup 2-6, 5-1, 5-3, 5-4
 - text 2-5
- filename 2-3, 6-2
- fill portion of screen B-9
- flood fill B-9, B-10
- floppy disk names 2-12
- fonts 5-2
- foreground color B-7, B-8
- fork 3-7, 4-6
 - request 4-3
- FORMAT 6-46
- FREE 6-49

- generate messages 6-42
- GET 2-6
- getting started, editor 7-4
- graphic window fonts 5-2
- graphics B-1
 - color set B-7
 - command codes B-7
 - cursor B-6, B-8
 - mode, select B-10
 - end B-8
 - erase B-8
 - medium resolution B-6
 - VDG B-6
- group 2-7
- grouping, commands 3-9

- halt data output 7-3
- hardware 1-2
- header
 - information 6-52
 - module 2-7, 3-3, 4-7
- HELP 6-51
- hex 6-2
- hexadecimal code display 6-38
- home
 - alpha cursor B-9
 - cursor B-4
- hours 6-2

- I-code 3-13
- I/O
 - paths 3-4
 - transfers 3-8
 - raw 3-8
- ID, process 4-4
- IDENT 3-3, 6-52
- images, pointer 5-2
- immortal
 - process 6-91
 - shell 3-11
- INIT 5-1
- initialize
 - a disk 6-46
 - a window 6-103
- INIZ 6-55
- input 2-1
 - lines 3-12
 - path 3-4
 - redirect 6-91
 - standard 4-4
- insert lines, editor 7-10
- interpreter, commands 6-90
- interprocess communication 3-7
- interrupt editor 7-3
- IOMAN 1-2, 1-3, 5-1

- kernel 1-1, 1-2
- keyboard 1-1
- keyword 3-1 - 3-3
- KILL 6-56
- kill 3-12
 - a directory 6-33, 6-33
 - files 6-31

- length
 - of video page 6-94
 - word 5-5, 5-6, 6-96, 6-109

- line
 - backspace 6-93, 6-106
 - delete, editor 7-3
 - draw B-8, B-10
 - duplicate 6-95

-
- line (*cont'd*)
 - duplication 6-107
 - erase B-4, B-8, B-9, B-10
 - syntax 6-1
 - linefeed 6-94, 6-107
 - lines, command 3-1
 - LINK 6-58
 - LIST 2-3, 2-5, 2-8, 3-4, 6-59
 - list
 - a directory 6-35
 - current processes 6-80
 - memory module names 6-64
 - segment 2-5
 - with program files 2-8
 - LOAD 4-7, 4-7, 6-61
 - lock a module 6-58
 - lockout 2-11
 - logical sector 2-3, 2-4
 - lowercase 6-93, 6-106
 - characters B-2
 - machine language 3-12
 - macro text editor 7-1
 - macros, edit 7-25
 - MAKDIR 2-11, 2-3, 6-63
 - management, memory 4-5
 - manager
 - pipe 1-2
 - random block 1-2
 - mark space 6-95, 6-108
 - MDIR 6-64
 - MDM kill 5-6, 5-7
 - medium resolution graphics B-6
 - memory
 - address 4-5
 - allocation 3-1
 - display free 6-69
 - load a file into 6-61
 - management 1-1, 4-5
 - size modifier 3-3
 - memory modules
 - lock 6-58
 - modules (*cont'd*)
 - unlink 6-100
 - deleting 4-7
 - display names 6-64
 - MERGE 6-68
 - messages with ECHO 6-42
 - messages, error 6-43
 - MFREE 6-69
 - minutes 6-2
 - MMU 4-5
 - mode, alpha B-3
 - mode
 - alphanumeric B-1
 - concurrent 3-10
 - semigraphic B-1
 - modem 1-1, 5-6, 5-7
 - auto-answer 6-108
 - name 2-12
 - modifier 3-1 - 3-3
 - execution 3-1, 3-3
 - memory size 3-3
 - redirection 3-5
 - modname 6-2
 - MODPAK 5-7
 - MODPATCH 6-70, 6-71, 6-72, 6-73
 - module 1-3
 - deleting memory 4-7
 - directory 4-6
 - header 2-7, 3-3, 4-7
 - header information 6-52
 - loading 4-7
 - lock in memory 6-58
 - primary 4-3
 - program 2-7
 - unlink 4-8
 - month 6-2
 - MONTYPE 6-74, 6-75
 - move cursor B-4, B-5, B-10
 - multiprogramming 4-1
 - multitasking 1-1
 - name
 - device 2-12, 2-13
 - modem 2-12
-

- name (*cont'd*)
 - printer 2-12
 - program 3-3
 - terminal 2-12
- next line, display 7-2
- not ready delay 5-6
- notations, syntax 6-1
- null count 6-94, 6-107
- number
 - priority 3-12
 - user 2-8, 4-4
- numeric parameters, editor 7-3
- object code 2-7
- operating system 1-3
- opts 6-2
- OS9Boot 5-1, 5-4
 - create 6-16, 6-76
- OS9Gen 5-2, 5-3, 6-72, 6-76
- OS9p2 5-1
- output 2-1
 - error 6-91
 - path 3-4, 4-4
 - redirect 3-11, 6-91
- owner 2-5, 2-8
- page length, video 6-107
- pages 3-3
- paint B-9
- parameter 3-1, 4-4
 - change system 6-93
 - command editor 7-3
- paramlist 6-2
- parent process 4-2
- parity 5-6, 5-7, 6-95, 6-108
- passing data 4-4
- pathlist 6-2
- paths 2-1
 - I/O 3-4
 - output 4-4
 - standard 3-4, 6-93
- patterns, semigraphic B-2
- pause 6-94
 - character 6-95, 6-108
 - screen 6-107
- permission 6-2
 - execute 2-9, 2-10
 - read 2-9, 2-10
 - write 2-9, 2-10
- physical sector 2-4
- PIC 4-8
- pipe 1-2, 3-7, 5-1
- pipelines 3-7, 3-8
- PIPEMAN 5-1
- Piper 5-1
- point
 - erase B-9
 - set B-8, B-10
- pointer
 - edit 7-1, 7-2
 - editor 7-7
 - file 2-4
 - images 5-2
- port 1-2
- port, RS-232 5-4, 6-109
- position alpha cursor B-9
- position-independent 2-7
 - code 4-8
- prepare a disk 6-46
- preset screen B-7
- previous line repeat 7-2
- primary module 4-3
- PRINTER 5-1
- printer 1-1, 1-2
 - name 2-12
 - test 6-98
- priority
 - number 3-12
 - change 6-88
 - process 4-2, 4-4
- procedure file 2-6, 3-10, 3-11
- process
 - background 3-7
 - chaining 6-44
 - child 3-6
 - create 3-6
 - current 6-80

-
- process (*cont'd*)
 - data 2-1
 - fork 3-7
 - ID 4-4
 - memory size 6-91
 - priority 4-2, 4-4
 - properties 4-4
 - sibling 4-3
 - state 4-2
 - terminate 6-56
 - time sharing 4-1
 - processor time 4-1
 - procID 6-2
 - PROCS 3-7, 4-2, 6-80
 - program
 - application 1-3
 - chaining 6-44
 - comments in 3-12
 - execution 6-90
 - modules 2-7
 - name 3-3
 - size 3-3
 - prompt 3-12
 - prompting 6-92
 - properties, process 4-4
 - public 2-9, 2-10
 - PUT 2-6
 - PWD 6-82
 - PXD 6-82

 - quick reference, editor 7-55
 - quit character 6-95, 6-108

 - RAM 4-5
 - random access 1-2
 - files 2-6
 - random block file manager 1-2
 - rate, baud 5-4, 5-5, 5-6, 6-96, 6-98, 6-109
 - raw I/O 3-8
 - RBF 5-1
 - read 2-1, 2-11, 2-4
 - permission 2-10, 2-9
 - readers 3-8

 - record 2-2, 2-6
 - lockout 2-11
 - redirect
 - data 3-4
 - input 6-91
 - output 6-91
 - redirection 3-1
 - modifiers 3-4, 3-5
 - output 3-11
 - symbols 3-5
 - re-entrant code 4-6
 - remove
 - directory 6-33
 - files 6-31
 - memory module 6-100
 - RENAME 6-84
 - repeat previous line 7-2
 - reprint character 6-95, 6-107
 - reserved characters 3-3
 - ROOT 2-2, 2-3
 - route data 3-4
 - RS-232 5-1, 5-4, 6-96, 6-109
 - run-time module 3-12
 - RUNB 3-13

 - SAVE 6-72
 - SCF 5-1
 - screen
 - alpha B-5
 - background 5-2
 - clear B-5
 - control B-1
 - pause 6-94, 6-107
 - preset B-7
 - scroll pause 6-94, 6-107
 - searching, editor 7-13
 - secondary buffer 7-1
 - seconds 6-2
 - sector 2-4
 - copy 6-7
 - displayed unused 6-49
 - file descriptor 2-5
 - logical 2-3
-

- security
 - file 2-8
 - permission 6-5
- SEEK 2-6
- segment list 2-5
- select
 - alpha mode B-10
 - color B-10
 - graphics mode B-10
- semicolon, sequential
 - execution 3-6
- semigraphic
 - mode B-1
 - patterns B-2
- sending data 2-1
- separator 3-1
 - ampersand 3-6
 - command 3-5
 - exclamation mark 3-8
- sequential execution 3-5, 3-6, 6-91
- set a window 6-103
- set a point B-8, B-10
- set priority 3-12
- SETIME 5-3, 6-86
- SETPR 6-88
- share time 4-1
- shell 1-3, 3-1-3-3, 3-8, 6-3
- SHELL 3-6, 6-90
- show
 - a directory 6-35
 - error message 6-43
 - execution directory 6-82
 - file contents 6-59
 - free memory 6-69
 - header information 6-52
 - memory module
 - names 6-64
 - working directory 6-82
- sibling processes 4-3
- sign bit 2-2
- simultaneous execution 3-5
- single-user file 2-8
- SIO 5-1
- size
 - file 2-5
 - process memory 6-91
 - program 3-3
- slash in device names 2-13
- sleeping 4-3
- software fonts 5-2
- sound bell B-10
- standard input 4-4, 6-93
- standard paths 3-4, 6-93
- start a window 6-103
- Startup 2-2, 2-6, 5-1, 5-3, 5-4, 6-75
- state 4-2, 4-2
- Stdfonts 5-2
- Stdpaths 5-2
- Stdptrs 5-2
- stop bit 5-5, 5-6
- string parameters, editor 7-4
- subdirectory 2-3
 - delete 6-33
- submanager 1-2
- subshell 3-10
- substituting, editor 7-13
- summary, commands 6-3, 6-4
- super user 6-56
- switch screen B-5
- symbols, redirection 3-5
- syntax 6-1
- SYS directory 5-1, 5-4
- system
 - administrator 1-1
 - date 6-86
 - disk create 5-3, 5-4
 - parameters 6-93
 - priority 6-88
 - time 6-86
- system diskette 2-2
 - create 6-16, 6-18, 6-76
- task, background 4-1
- term 1-1
- TERM 5-1
- TERM-VDG 6-96
- TERM-WIN 6-109

- terminal name 2-12
 - terminals 1-2
 - terminate
 - a character 6-95, 6-108
 - a process 6-56
 - the editor 7-2
 - on error 6-92
 - test delay loop 6-98
 - text 6-2, B-2
 - buffers 7-1
 - display, editor 7-6
 - editing 7-1
 - file operations, editor 7-18
 - files 2-5
 - tick 4-1, 4-2
 - tickcount 6-2
 - time 2-5, 6-24
 - sharing, process 4-1
 - CPU 4-1
 - processor 4-1
 - set 6-86
 - timeslice 4-1, 4-2
 - TMODE 6-93
 - tracks 2-5
 - transfer, I/O 3-8
 - transferring data 3-7
 - TUNEPORT 6-98
 - turn on
 - cursor B-5
 - echo 6-92
 - prompting 6-92
 - type 5-7
 - ACIA 6-96, 6-108
 - of window 6-103
 - value 5-7

 - UNLINK 4-7, 4-8, 6-100
 - unused disk sectors 6-49
 - update mode 2-11
 - uppercase 6-93, 6-106
 - characters B-2

 - user
 - bit 2-11
 - number 2-8, 4-4
 - value 6-2
 - type 5-7
 - variable 6-1
 - VDG graphics B-6
 - video 1-1
 - page length 6-94, 6-107
 - view
 - current processes 6-80
 - error messages 6-43
 - working directory 6-82

 - waiting state 4-3
 - WCREATE 6-103
 - window 5-2
 - alpha mode controls B-3
 - descriptor 2-12
 - initialization 6-103
 - type 6-103
 - word length 5-5, 5-6, 6-96, 6-109
 - working directory 6-12
 - write 2-1, 2-4, 2-11
 - permission 2-9, 2-10

 - XMODE 5-4, 5-5, 6-106

 - year 6-2
-

Technical Reference

OS-9
Technical
Reference

Contents

Chapter 1 System Organization	1-1
I/O System Modules	1-1
Color Computer OS-9 Modules	1-2
Kernel, Clock Module, and INIT	1-2
Input/Output Modules	1-3
I/O Manager	1-3
File Managers	1-3
Device Drivers	1-3
Device Descriptors	1-4
Shell	1-4
Chapter 2 The Kernel	2-1
System Initialization	2-1
System Call Processing	2-4
OS9Defs and Symbolic Names	2-4
Types of System Calls	2-4
Memory Management	2-5
Memory Use	2-5
Color Computer OS-9 Typical Memory Map	2-7
Memory Management Hardware	2-7
Multiprogramming	2-12
Process Creation	2-12
Process States	2-13
Execution Scheduling	2-14
Signals	2-15
Interrupt Processing	2-16
Logical Interrupt Polling System	2-17
Virtual Interrupt Processing	2-19
Chapter 3 Memory Modules	3-1
Module Types	3-1
Module Format	3-1
Module Header	3-2
Module Body	3-2
CRC Value	3-2
Module Headers: Standard Information	3-3
Sync Bytes	3-3
Module Size	3-3
Offset to Module Name	3-3
Type/Language Byte	3-4
Attributes/Revision Level Byte	3-4
Header Check	3-5

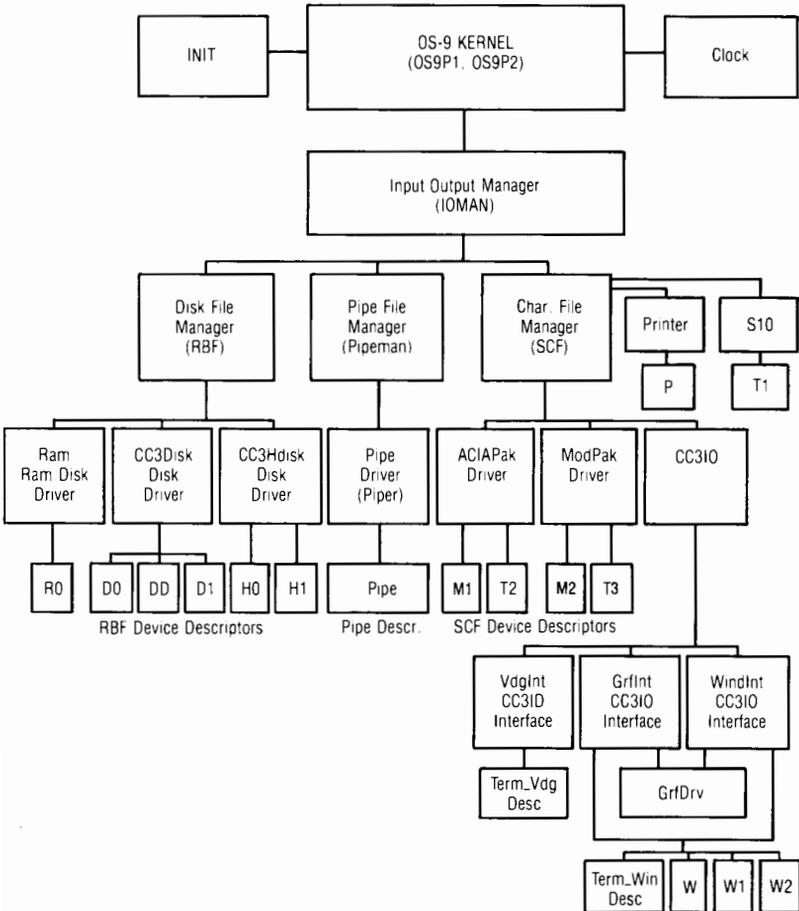
Module Headers: Type-Dependent Information	3-5
Executable Memory Module Format	3-6
Chapter 4 OS-9's Unified Input/Output System	4-1
I/O System Modules	4-1
The I/O Manager	4-2
File Managers	4-3
File Manager Structure	4-3
Create, Open	4-4
Mkdir	4-4
ChgDir	4-4
Delete	4-5
Seek	4-5
Read	4-5
Write	4-6
ReadLn	4-6
WriteLn	4-6
GetStat, PutStat	4-6
Close	4-7
Interfacing with Device Drivers	4-7
Device Driver Modules	4-8
Device Driver Module Format	4-10
OS-9 Interaction With Devices	4-11
Suspend State (Level Two Only)	4-13
Device Descriptor Modules	4-15
Path Descriptors	4-18
Chapter 5 Random Block File Manager	5-1
Logical and Physical Disk Organization	5-1
Identification Sector (LSN 0)	5-2
Disk Allocation Map Sector (LSN 1)	5-3
ROOT Directory	5-3
File Descriptor Sector	5-3
Directories	5-5
The RBF Manager Definitions of the Path Descriptor ..	5-5
RBF-Type Device Descriptor Modules	5-8
RBF Record Locking	5-10
Record Locking and Unlocking	5-11
Non-Shareable Files	5-12
End-of-File Lock	5-12
Deadlock Detection	5-13
RBF-Type Device Driver Modules	5-13
The RBF Device Memory Area Definitions	5-13
RBF Device Driver Subroutines	5-16

Chapter 6 Sequential Character File Manager	6-1
SCF Line Editing Functions	6-1
Read and Write	6-1
Read Line and Write Line	6-2
SCF Definitions of the Path Descriptor	6-2
SCF-Type Device Descriptor Modules	6-6
SCF-Type Device Driver Modules	6-9
SCF Device Driver Subroutines	6-10
Chapter 7 The Pipe File Manager (PEPEMAN)	7-1
Chapter 8 System Calls	8-1
Calling Procedure	8-1
I/O System Calls	8-2
System Call Descriptions	8-2
User Mode System Calls Quick Reference	8-3
System Mode Calls Quick Reference	8-5
User System Calls	8-7
I/O User System Calls	8-44
Privileged System Mode Calls	8-66
Get Status System Calls	8-112
Set Status System Calls	8-130
Appendices	A-1
A Memory Module Diagrams	A-1
B Standard Floppy Disk Format	B-1
C System Error Codes	C-1
Index	

System Organization

OS-9 is composed of a group of modules, each of which has a specific function. The following illustration shows the major modules. Actual module names are capitalized.

I/O System Modules



OS-9 COMPONENT MODULE ORGANIZATION

Color Computer OS-9 Modules

IOMAN	Input/output management
INIT	System initialization table
CLOCK	Software routine time management
RBF	Random block file management
SCF	Sequential character file management
PIPEMAN	Pipe file management
CC3DISK	Color Computer disk driver
CC3IO	Color Computer input/output driver

The VDGINT (video display generator) provides both interface functions and low-level routines for Color Computer 2 VDG compatibility.

The GRFINT interface provides high-level graphics code interpretation and interface functions.

The WINDINT interface contains all the functions of GRFINT, along with additional support for Multiview functions. If you are using Multiview, exclude GRFINT from the system.

Both WINDINT and GRFINT use the low-level driver GRFDRV to perform the actual drawing on bitmap screens.

Term_VDG uses CC3IO and VDGINT. TERM_WIN and all window descriptors (W, W1, W2, and so on) use CC3IO, WINDINT, GRFINT, and GRFDRV modules.

Kernel, Clock Module, and INIT

The system's first level contains the *kernel*, *clock module*, and *INIT*.

The kernel provides basic system services, such as multitasking and memory management. It links all other OS-9 modules into the system.

The clock module is a software handler for the real-time clock hardware.

INIT is an initialization table used by the kernel during system startup. This table loads initial tasks and specifies initial table sizes and initial system device names. It is loaded into RAM (random access memory) by the OS-9 bootstrap module Boot. Boot also loads the OS9P2 and INIT modules during system startup.

There are two ways to run boot:

- Using the DOS command with Color Disk BASIC, Version 1.1, or later.
- Pressing the reset button after OS-9 is running.

Input/Output Modules

The remaining modules make up the OS-9 I/O system. They are defined briefly here and are discussed in detail in Chapter 4.

I/O Manager

The system's second level (the level below the kernel) contains the input/output manager, IOMAN. The I/O manager provides common processing for all input/output operations. It is required for performing any input/output supported by OS-9.

File Managers

The system's third level contains the *file managers*. File managers perform I/O request processing for similar classes of I/O devices. There are three file managers:

- | | |
|--------------------|--|
| RBF manager | The random block file manager processes all disk I/O operations. |
| SCF manager | The sequential character file manager handles all non-disk I/O operations that operate one character at a time. These operations include terminal and printer I/O. |
| PIPEMAN | The pipe file manager handles <i>pipes</i> . Pipes are memory buffers that act as files. Pipes are used for data transfers between processes. |

Device Drivers

The system's fourth level contains the *device drivers*. Device drivers handle basic I/O functions for specific I/O controller hardware. You can use pre-written drivers, or you can write your own.

Device Descriptors

The system's fifth level contains the *device descriptors*. Device descriptors are small tables that define the logical name, device driver, and file manager for each I/O port. They also contain port initialization and port address information. Device descriptors require only one copy of each I/O controller driver used.

Shell

The shell is the command interpreter. It is a program and not a part of the operating system. The shell is fully described in the *OS-9 Commands* manual.

The Kernel

The kernel is the core of OS-9. The kernel supervises the system and manages system resources. Half of the kernel (called OS9P1) resides in the boot module. The other half of the kernel (called OS9P2) is loaded into RAM with the other OS-9 modules.

The kernel's main functions are:

- System initialization after reset
- Service request processing
- Memory management
- Multiprogramming management
- Interrupt processing

I/O functions are not included in the list because the kernel does not directly process them. Instead, it passes I/O system calls to the I/O Manager for processing.

System Initialization

After a hardware reset, the kernel initializes the system. This involves:

1. Locating modules loaded in memory from the OS-9 Boot file.
2. Determining the amount of available RAM.
3. Loading any required modules that were not loaded from the OS-9 Boot file.

OS-9 Level Two cannot install new system calls using the OS-9 Level One system call F\$SSvc. F\$SSvc does not work with a Level Two user program because of the separation of system and user address space.

OS-9 Technical Reference

OS9P3 can be used to tailor the system to fit specific needs. The following listing is an example of how to use the OS9P3 module.

Microware OS-9 Assembler 2.1 11/18/83 16:06:01 Page 001

OS-9 Level TWO v1.2, part 2 - OS-9 System Symbol Definitions

```
00001
00002
00003

00011 *****
00012 *
00013 *           Module Header
00014 *
00015 00C1           Type      set   Systm*Objct
00016 00B1           Revs     set   ReEnt+1
00017 0000 07CD005E      mod   OS9End,OS9Name,Type,Revs,Cold,256
00018 000D 4F533970    OS9Name fcs   "OS9p3"
00019
00029 0012 01           fcb   !  edition number
00030           use   defsfile
00031 0002           level   equ   2
00032           opt   -c
00033           opt   f
00041
00042 *****
00043 *
00044 *           Routine Cold
00045 *
00046 *
00047
00048 0013 318D0004      Cold   leay  SvcTbl,pcr  get service routine
00049 0017 103F32           DS9   F$SSvc  install new service
00050 001A 39             rts
00051
00052
00053 *****
00054 *
00055 *           Service Routines Initialization Table
00056 *
00057
00058 0025           F$SAYHI  equ   $25 set up new call
00059 *           Add this to the user os9defs file.
00060
```

```

00061 001B          SvcTbl  equ  *
00062 001B 25          fcb  F$SAYH!
00063 001C 0001       fdb  SayHi-**-2
00064 001E 00        fcb  $80

```

Microware OS-9 Assembler 2.1 11/18/83 16:06:01 Page 002
 OS-9 Level TWO V1.2, part 2 - OS-9 System Symbol Definitions

```

00068 *
00069 *Service call Say Hello to user
00070 *
00071 *Input: U = Registers ptr
00072 *          R$X,u = Message ptr (if 0 send default)
00073 *          Max message length = 40 bytes.
00074 *
00075 *Output: Message sent to standard error path of user.
00076 *
00077 *Data: D.Proc
00078 *
00079
00080 001F AE44 SayHi ldx R$X,u get mess. address
00081 0021 2619     bne SayHi6 bra if not default
00082 0023 109ES0   ldy D.Proc get proc desc ptr
00083 0026 EE24     ldu P$SP,y get: caller's stack
00084 0028 33C8D8   leau -40,u room for message
00085 002B 96D0     lda D.SysTsk system's task num
00086 002D E626     ldb P$TASK,y caller's task num
00087 002F 108E0028 ldy #40 set byte count
00088 0033 308D0012 leax Hello,pcr destination ptr
00089 0037 103F38   DS9 F$Move mess into user mem
00090 003A 30C4     leax 0,u
00091 003C 108E0028 SayHi6 ldy #40 get: max byte count
00092 0040 DE50     ldu D.Proc get proc desc ptr
00093 0042 A6C832   lda P$PATH+2,u path num of stderr
00094 0045 103F8C   DS9 !$Writ_n write mess line
00095 0048 39      rts
00096
00097 0049 48656C6C Hello fcc "Hello there user."
00098 005A 0D      fcb  $D
00099
00100 005B 5104B6   emod module CRC
00101

```

```
00102 005E      OS9End   equ    *
00103
00104                               end

00000 error(s)
00000 warning(s)
$005E 00094 program bytes generated
$0000 00000 data bytes allocated
$2884 10372 bytes used for symbols
```

System Call Processing

System calls are used to communicate between OS-9 and assembly-language programs for such functions as memory allocation and process creation. In addition to I/O and memory management functions, system calls have other functions. These include interprocess control and timekeeping.

System calls use the SWI2 instruction followed by a constant byte representing the code. You usually pass parameters for system calls in the 6809 registers.

OS9Defs and Symbolic Names

A system-wide assembly-language *equate file*, called OS9Defs, defines symbolic names for all system calls. This file is included when assembling hand-written or compiler-generated code. The OS-9 assembler has a built-in *macro* to generate system calls. For example:

```
OS9 I$Read
```

is recognized and assembled as equivalent to:

```
SWI2
FCB I$Read
```

The OS-9 assembly macro OS9 generates an SWI2 function. The label I\$Read is the label for the code \$89.

Types of System Calls

System calls are divided into two categories, *I/O calls* and *function calls*.

I/O calls perform various input/output functions. The kernel passes calls of this type to the I/O manager for processing. The symbolic names for I/O calls begin with I\$. For example, the Read system call is called I\$Read.

Function calls perform memory management, multi-programming, and other functions. Most are processed by the kernel. The symbolic names for function calls begin with F\$. For example, the Link function call is called F\$Link.

The function calls include *user calls* and *privileged system mode* calls. (See Chapter 8, "System Calls", for more information.)

Memory Management

Memory management is an important operating system function. Using memory modules, OS-9 manages the logical contents of memory and the physical assignment of memory to programs.

All programs that are loaded must be in the memory module format. This format allows OS-9 to maintain a module directory of all the programs in memory. The directory contains information about each module, including its name and address and the number of processes using it. The number of processes using a module is called the module's *link count*.

When a module's link count is zero, OS-9 deallocates its part of memory and removes its name from the module directory.

Memory modules are the foundation of OS-9's modular software environment. Advantages of memory management are:

- Automatic runtime linking of programs to libraries of utility modules
- Automatic sharing of re-entrant programs
- Replacement of small sections of large programs into memory for update or correction

Memory Use

OS-9 reserves some space at the top and bottom of RAM for its own use. The amount depends on the sizes of system tables that are specified in the INIT module.

OS-9 pools all other RAM into a free memory space. As the system allocates or deallocates memory, it dynamically takes it from or returns it to this pool. RAM does not need to be contiguous because the memory management unit can dynamically rearrange memory addresses.

The basic unit of memory allocation is the 256-byte *page*. OS-9 always allocates memory in whole numbers of pages.

The data structure that OS-9 Level Two uses to keep track of memory allocation is a 256-byte *bit map*. Each bit in this table is associated with a specific page of memory. A cleared bit indicates that the page is free and available for assignment. A set bit indicates that the page is in use (that no RAM is free at that address). OS-9 Level Two always allocates memory in 8192-byte increments. This is the smallest memory block that the memory management hardware supports.

OS-9 automatically allocates memory when any of the following occurs:

- Program modules are loaded into RAM
- Processes are created
- Processes execute system calls to request additional RAM
- OS-9 needs I/O buffers or larger tables

OS-9 also has inverse functions to deallocate memory allocated to program modules, new processes, buffers, and tables.

In general, memory for program modules and buffers is allocated from high addresses downward. Memory for process data areas is allocated from low addresses upward.

Following, is a memory map of a typical system. Actual memory sizes and addresses can vary depending on the exact system configuration.

Color Computer OS-9 Typical Memory Map

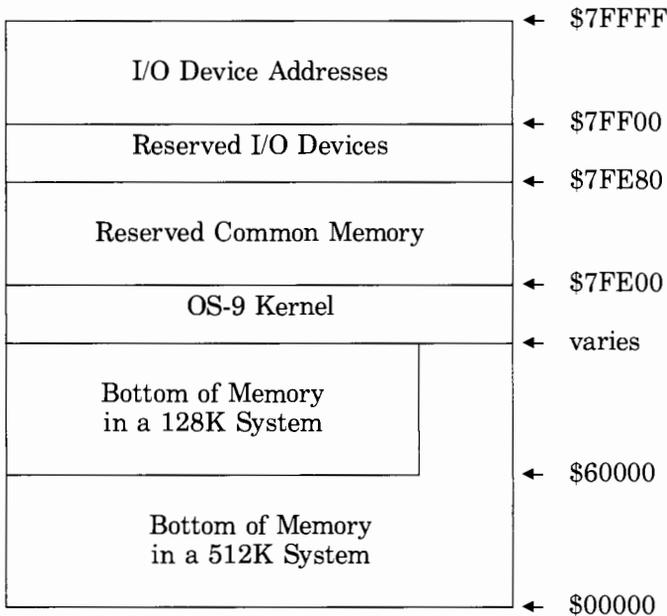


Figure 2.1

Note: The high two pages of every logical address space contain the defined areas I/O Device Addresses, Reserved I/O Devices, and Reserved Common Memory.

Memory Management Hardware

The 8-bit CPU in the Color Computer 3 can directly address only 64 kilobytes of memory using its 16 address lines (A0-A15). The Color Computer 3's Memory Management Unit (MMU) extends the addressing capability of the computer by increasing the address lines to 19 (A0-A18). This lets the computer address up to 512 kilobytes of memory (\$0-\$7FFFF).

The 512K address space is called the *physical address space*. The physical address space is subdivided into 8K *blocks*. The six high order address bits (A13-A18) define a *block number*.

OS-9 creates a *logical address space* of up to 64K for each task by using the FORK system call. Even though the memory within a logical address space appears to be contiguous, it might not be—the MMU translates the physical addresses to access available memory. Address spaces can also contain blocks of memory that are common to more than one map.

The MMU consists of a multiplexer and a 16 by 6-bit RAM array. Each of the 6-bit elements in this array is an MMU task register. The computer uses these task registers to determine the proper 8-kilobyte memory segment to address.

The MMU task registers are loaded with addressing data by the CPU. This data indicates the actual location of each 8-kilobyte segment of the current system memory. The task registers are divided into two sets consisting of eight registers each. Whether the task register select bit (TR bit) is set or reset, determines which of the two sets is to be used.

The relation between the data in the task register and the generated addresses is as follows:

Bit	D5	D4	D3	D2	D1	D0
Corresponding Memory Address	A18	A17	A16	A15	A14	A13

Figure 2.2

When the CPU accesses any memory outside the I/O and control range (XFF00=XFFFF), the CPU address lines (A13-A15) and the TR bit determine what segment of memory to address. This is done through the multiplexer when SELECT is low. (See the following table.)

When the CPU writes data to the MMU, A0-A3 determine the location of the MMU register to receive the incoming data when SELECT is high. The following diagram illustrates the operation of the Color Computer 3's memory management:

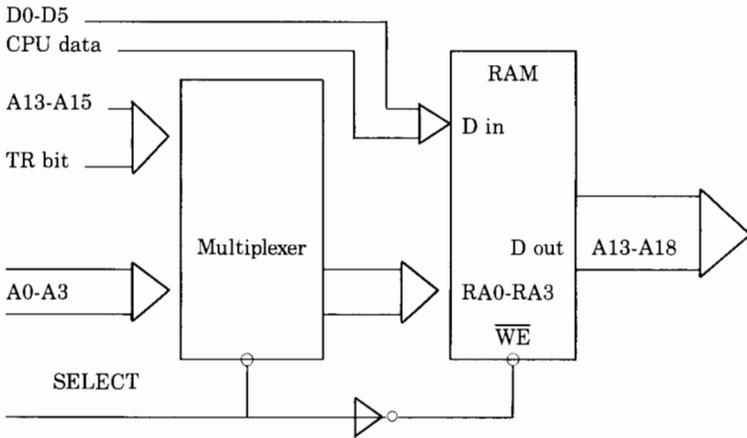


Figure 2.3

The system uses the data from the MMU registers to determine the block of memory to be accessed, according to the following table:

TR Bit	A15	A14	A13	AddressRange	MMU Address
0	0	0	0	X0000-X1FFF	FFA0
0	0	0	1	X2000-X3FFF	FFA1
0	0	1	0	X4000-X5FFF	FFA2
0	0	1	1	X6000-X7FFF	FFA3
0	1	0	0	X8000-X9FFF	FFA4
0	1	0	1	XA000-XBFFF	FFA5
0	1	1	0	XC000-XDFFF	FFA6
0	1	1	1	XE000-XFFFF	FFA7
1	0	0	0	X0000-X1FFF	FFA8
1	0	0	1	X2000-X3FFF	FFA9
1	0	1	0	X4000-X5FFF	FFAA
1	0	1	1	X6000-X7FFF	FFAB
1	1	0	0	X8000-X9FFF	FFAC
1	1	0	1	XA000-XBFFF	FFAD
1	1	1	0	XC000-XDFFF	FFAE
1	1	1	1	XE000-XFFFF	FFAF

Figure 2.4

The translation of physical address to 8K-blocks is as follows:

Range		Block Number	Range		Block Number
From	To		From	To	
00000	01FFF	00	40000	41FFF	20
02000	03FFF	01	42000	43FFF	21
04000	05FFF	02	44000	45FFF	22
06000	07FFF	03	46000	47FFF	23
08000	09FFF	04	48000	49FFF	24
0A000	0BFFF	05	4A000	4BFFF	25
0C000	0DFFF	06	4C000	4DFFF	26
0E000	0FFFF	07	4E000	4FFFF	27
10000	11FFF	08	50000	51FFF	28
12000	13FFF	09	52000	53FFF	29
14000	15FFF	0A	54000	55FFF	2A
16000	17FFF	0B	56000	57FFF	2B
18000	19FFF	0C	58000	59FFF	2C
1A000	1BFFF	0D	5A000	5BFFF	2D
1C000	1DFFF	0E	5C000	5DFFF	2E
1E000	1FFFF	0F	5E000	5FFFF	2F
20000	21FFF	10	60000	61FFF	30
22000	23FFF	11	62000	63FFF	31
24000	25FFF	12	64000	65FFF	32
26000	27FFF	13	66000	67FFF	33
28000	29FFF	14	68000	69FFF	34
2A000	2BFFF	15	6A000	6BFFF	35
2C000	2DFFF	16	6C000	6DFFF	36
2E000	2FFFF	17	6E000	6FFFF	37
30000	31FFF	18	70000	71FFF	38
32000	33FFF	19	72000	73FFF	39
34000	35FFF	1A	74000	75FFF	3A
36000	37FFF	1B	76000	77FFF	3B
38000	39FFF	1C	78000	79FFF	3C
3A000	3BFFF	1D	7A000	7BFFF	3D
3C000	3DFFF	1E	7C000	7DFFF	3E
3E000	3FFFF	1F	7E000	7FFFF	3F

Figure 2.5

In order for the MMU to function, the TR bit at \$FF90 must be cleared and the MMU must be enabled. However, before doing this, the address data for each memory segment must be loaded into the designated set of task registers. For example, to select a standard 64K map in the top range of the Color Computer 3's 512K RAM, with the TR bit set to 0, the following values must be preloaded into the MMU's registers:

MMU Location Address	Data (Hex)	Data (Bin)	Address Range
FFA0	38	111000	70000-71FFF
FFA1	39	111001	72000-73FFF
FFA2	3A	111010	74000-75FFF
FFA3	3B	111011	76000-77FFF
FFA4	3C	111100	78000-79FFF
FFA5	3D	111101	7A000-7BFFF
FFA6	3E	111110	7C000-7DFFF
FFA7	3F	111111	7E000-7FFFF

Figure 2.6

Although this table shows MMU data in the range \$38 to 3F, any data between \$0 and \$3F can be loaded into the MMU registers to select memory addresses in the range 0 to \$7FFFF, as illustrated by Figure 2.5.

Normally, the blocks containing I/O devices are kept in the system map, but not in the user maps. This is appropriate for time-sharing applications, but not for process control. To directly access I/O devices, use the F\$MspBlk system call. This call takes a starting block number and block count, and maps them into *unallocated* spaces of the process's address space. The system call returns the logical address at which the blocks were inserted.

For example, suppose a display screen in your system is allocated at extended addresses \$7A000-\$7DFFFF (blocks 3D and 3E). The following system call maps them into your address space:

```
ldb    #2          number of blocks
ldx    #3D         starting block number
os9    F$MapBlk   call MapBlk
stu    IOPorts    save address where mapped
```

On return, the U register contains the starting address at which the blocks were switched. For example, suppose that the call returned \$4000. To access extended address \$7A020, write to \$4020.

Other system calls that copy data to or from one task's map to another are available, such as F\$STABX and F\$Move. Some of these calls are system mode privileged. You can unprotect them by changing the appropriate bit in the corresponding entry of the system service request table and then making a new system boot with the patched table.

Multiprogramming

OS-9 is a multiprogramming operating system. This means that several independent programs called *processes* can be executed at the same time. By issuing the appropriate system call to OS-9, each process can have access to any system resource.

Multiprogramming functions use a hardware real-time clock. The clock generates interrupts 60 times per second, or one every 16.67 milliseconds. These interrupts are called ticks.

Processes that are not waiting for some event are called *active processes*. OS-9 runs active processes for a specific system-assigned period called a time slice. The number of time slices per minute during which a process is allowed to execute depends on a process's priority relative to all other active processes. Many OS-9 system calls are available to create, terminate, and control processes.

Process Creation

A process is created when an existing process executes a Fork system call (F\$Fork). This call's main argument is the name of the program module that the new process is to execute first (the *primary module*).

Finding the Module. OS-9 first attempts to find the module in the module directory. If it does not find the module, OS-9 usually attempts to load into memory a mass-storage file in the execution directory, with the requested module name as a filename.

Assigning a Process Descriptor. Once OS-9 finds the module, it assigns the process a data structure called a *process descriptor*. This is a 64-byte package that contains information about the process, its state (see the following section “Process States”), memory allocations, priority, queue pointers, and so on. OS-9 automatically initializes and maintains the process descriptor. The process itself cannot access the descriptor; it has no need to do so.

Allocate RAM. The next step is to allocate RAM for the process. The primary module’s header contains a storage size. OS-9 uses this size unless the Fork system call requests a larger area. OS-9 then attempts to allocate a memory area of the specified size from the free memory space. The memory space does not need to be contiguous.

Proceed or Terminate. If OS-9 can perform all of the previous steps, it adds the new process to the active process queue for execution scheduling. If it cannot, it terminates the creation; the process that originated the Fork is informed of the error.

Assign Process ID and User ID. OS-9 assigns the new process a unique number called a *process ID*. Other processes can communicate with the process by referring to its ID in various system calls.

The process also has a *user ID*, which is used to identify all processes and files belonging to a particular user. The user ID is inherited from the parent process.

Process Termination. A process terminates when it executes an Exit system call (F\$Exit) or when it receives a *fatal* signal. The termination closes any open paths, deallocates memory used by the process, and unlinks its primary module.

Process States

At any instant a process can be in one of three states:

- Active—The process is ready for execution.
- Waiting—The process is suspended until a *child process* terminates or until it receives a signal. A child process is a process that is started (execution is begun by) another process—a *parent process*.

- **Sleeping**—The process is suspended for a specific period of time or until it receives a signal.

Each state has its own queue, a linked list of *descriptors* of processes in that state. To change a process's state, move its descriptor to another queue.

The Active State. Each active process is given a time slice for execution, according to its priority. The scheduler in the kernel ensures that all active processes, even those of low priority, get some CPU time.

The Wait State. This state is entered when a process executes a Wait system call (F\$Wait). The process remains suspended until one of its *child* processes terminates or until it receives a *signal*. (See the "Signals" section later in this chapter.)

The Sleeping State. This state is entered when a process executes a Sleep system call (F\$Sleep), which specifies the number of ticks for which the process is to remain suspended. The process remains asleep until the specified time has elapsed or until it receives a wakeup signal.

Execution Scheduling

The OS-9 scheduler uses an algorithm that ensures that all active processes get some execution time.

All active processes are members of the *active process queue*, which is kept sorted by process *age*. Age is the number of process switches that have occurred since the process's last time slice. When a process is moved to the active process queue from another queue, its age is set according to its priority—the higher the priority, the higher the age.

Whenever a new process becomes active, the ages of all other active processes increase by one time slice count. When the executing process's time slice has elapsed, the scheduler selects the next process to be executed (the one with the next highest age, the first one in the queue). At this time, the ages of all other active processes increase by one. Ages never go beyond 255.

A new active process that was terminated while in the system state is an exception. This process is given high priority because it is usually executing critical routines that affect shared system resources.

When there are no active processes, the kernel handles the next interrupt and then executes a CWA1 instruction. This procedure decreases interrupt latency time (the time it takes the system to process an interrupt).

Signals

A *signal* is an asynchronous control mechanism used for inter-process communication and control. It behaves like a software interrupt. It can cause a process to suspend a program, execute a specific routine, and then return to the interrupted program.

Signals can be sent from one process to another process by the Send system call (F\$Send). Or, they can be sent from OS-9 service routines to a process.

A signal can convey status information in the form of a 1-byte numeric value. Some *signal codes* (values) are predefined, but you can define most. The signal codes are:

0	= Kill (terminates the process, is non-interceptable)
1	= Wakeup (wakes up a sleeping process)
2	= Keyboard terminate
3	= Keyboard interrupt
4	= Window change
128-255	= User defined

When a signal is sent to a process, the signal is saved in the process descriptor. If the process is in the sleeping or waiting state, it is changed to the active state. When the process gets its next time slice, the signal is processed.

What happens next depends on whether or not the process has set up a *signal intercept trap* (signal service routine) by executing an Intercept system call (F\$Icpt).

If the process has set up a signal intercept trap, the process resumes execution at the address given in the Intercept call. The signal code passes to this routine. Terminate the routine with an RTI instruction to resume normal execution of the process.

Note: A wakeup signal activates a sleeping process. It sets a flag but ignores the call to branch to the intercept routine.

If it has not set up a signal intercept trap, the process is terminated immediately. It is also terminated if the signal code is zero. If the process is in the system mode, OS-9 defers the termination. The process dies upon return to the user state.

A process can have a signal pending (usually because the process has not been assigned a time slice since receiving the signal). If it does, and another process tries to send it another signal, the new signal is terminated, and the Send system call returns an error. To give the destination process time to process the pending signal, the sender needs to execute a Sleep system call for a few ticks before trying to send the signal again.

Interrupt Processing

Interrupt processing is another important function of the kernel. OS-9 sends each hardware interrupt to a specific address. This address, in turn, specifies the address of the device service routine to be executed. This is called *vectoring* the interrupt. The address that points to the routine is called the *vector*. It has the same name as the interrupt.

The SWI, SWI2, and SWI3 vectors point to routines that read the corresponding pseudo vector from the process's descriptor and dispatch to it. This is why the Set SWI system call (F\$SSWI) is local to a process; it only changes a pseudo vector in the process descriptor.

**Hardware Vector
Table**

<u>Vector</u>	<u>Address</u>
SWI3	\$FFF2
SWI2	\$FFF4
FIRQ	\$FFF6
IRQ	\$FFF8
SWI	\$FFFA
NMI	\$FFFC
RESTART	\$FFFE

FIRQ Interrupt. The system uses the FIRQ interrupt. The FIRQ vector is not available to you. The FIRQ vector is reserved for future use. Only one FIRQ generating device can be in the system at a time.

Logical Interrupt Polling System

Because most OS-9 I/O devices use IRQ interrupts, OS-9 includes a sophisticated polling system. The IRQ polling system automatically identifies the source of the interrupt, and then executes its associated user- or system-defined service routine.

IRQ Interrupt. Most OS-9 I/O devices generate IRQ interrupts. The IRQ vector points to the real-time clock and the keyboard scanner routines. These routines, in turn, jump to a special IRQ polling system that determines the source of the interrupt. The polling system is discussed in the next section, “Logical Interrupt Polling System.”

NMI Interrupt. The system uses the NMI interrupt. The NMI vector, which points to the disk driver interrupt service routine, is not available to you.

The Polling Table. The information required for IRQ polling is maintained in a data structure called the *IRQ polling table*. The table has a 9-byte entry for each device that might generate an IRQ interrupt. The table size is permanent and is defined by an initialization constant in the INIT module. Each entry in the polling table is given a number from 0 (lowest priority) to 255 (highest priority). In this way, the more important devices (those that have a higher interrupt frequency) can be polled before the less important ones.

Each entry has six variables:

- | | |
|------------------------|--|
| Polling Address | Points to the status register of the device. The register must have a bit or bits that indicate if it is the source of an interrupt. |
| Flip Byte | Selects whether the bits in the device status register indicate active when set or active when cleared. If a bit in the flip byte is set, it indicates that the task is active whenever the corresponding bit in the status register is clear. |

Mask Byte	Selects one or more interrupt request flag bits within the device status register. The bits identify the active task or device.
Service Routine Address	Points to the interrupt service routine for the device. You supply this address.
Static Storage Address	Points to the permanent storage area required by the device service routine. You supply this address.
Priority	Sets the order in which the devices are polled (a number from 0 to 255).

Polling the Entries. When an IRQ interrupt occurs, OS-9 enters the polling system via the corresponding RAM interrupt vector. It starts polling the devices in order of priority. OS-9 loads the status register address of each entry into Accumulator A, using the device address from the table.

OS-9 performs an exclusive-OR operation using the flip byte, followed by a logical-AND operation using the mask byte. If the result is non-zero, OS-9 assumes that the device is the source of the interrupt.

OS-9 reads the device memory address and service routine address from the table, and performs the interrupt service routine.

Note: If you are writing your own device driver, terminate the interrupt service routine with an RTS instruction, **not** an RTI instruction.

Adding Entries to the Table. You can make entries to the IRQ (interrupt request) polling table by using the Set IRQ system call (F\$IRQ). Set IRQ is a *privileged system call*, OS-9 can execute it only in the system mode. OS-9 is in system mode whenever it is running a device driver.

Note: The code for the interrupt polling system is located in the I/O Manager module. The OS9P1 and OS9P2 modules contain the physical interrupt processing routines.

Virtual Interrupt Processing

A virtual IRQ, or VIRQ, is useful with devices in Multi-Pak expansion slots. Because of the absence of an IRQ line from the Multi-Pak interface, these devices cannot initiate physical interrupts. VIRQ enables these devices to act as if they were interrupt driven. Use VIRQ only with device driver and pseudo device driver modules. VIRQ is handled in the Clock module, which handles the VIRQ polling table and installs the F\$VIRQ system call. Since the F\$VIRQ system call is dependent on clock initialization, the CC3GO module forces the clock to start.

The virtual interrupt is set up so that a device can be interrupted at a given number of clock ticks. The interrupt can occur one time, or can be repeated as long as the device is used.

The F\$VIRQ system call installs VIRQ in a table. This call requires specification of a 5-byte packet for use in the VIRQ table. This packet contains:

- Bytes for an actual counter
- A reset value for the counter
- A status byte that indicates whether a virtual interrupt has occurred and whether the VIRQ is to be re-installed in the table after being issued

F\$VIRQ also specifies an initial tick count for the interrupt. The actual call is summarized here and is described in detail in Chapter 8.

Call:	OS9 F\$VIRQ
Input:	(Y) = <i>address of 5-byte packet</i> (X) = 0 to delete entry, 1 to install entry (D) = <i>initial count value</i>
Output:	none (CC) carry set on error (IS) <i>appropriate error code</i>

The 5-byte packet is defined as follows:

Name	Offset	Function
Vi.Cnt	\$0	Actual counter
Vi.Rst	\$2	Reset value for counter
Vi.Stat	\$4	Status byte

Two of the bits in the status byte are used. These are:

Bit 0 - set if VIRQ occurs

Bit 7 - set if a count reset is required

When making an F\$VIRQ call, the packet might require initialization with a reset value. Bit 7 of the status byte must be either set or cleared to signify a reset of the counter or a one-time VIRQ call. The reset value does not need to be the same as the initial counter value. When OS-9 processes the call, it writes the packet address into the VIRQ table.

At each clock tick, OS-9 scans the VIRQ table and subtracts one from each timer value. When a timer count reaches zero, OS-9 performs the following actions:

1. Sets Bit 0 in the status byte. This specifies a Virtual IRQ.
2. Checks Bit 7 of the status byte for a count reset request.
3. If bit 7 is set, resets the count using the reset value. If bit 7 is reset, deletes the packet address from the VIRQ table.

When a counter reaches zero and makes a virtual interrupt request, OS-9 runs the standard interrupt polling routine and services the interrupt. Because of this, you must install entries on both the VIRQ and DIRQ polling tables whenever you are using a VIRQ.

Unless the device has an actual physical interrupt, install the device on the IRQ polling table via the F\$IRQ system call before placing it on the VIRQ table.

If the device has a physical interrupt, use the interrupt's hardware register address as the polling address for the F\$IRQ call. After setting the polling address, set the flip and mask bytes for the device, and make the F\$IRQ call.

If the device is totally VIRQ-driven, and has no interrupts, use the status byte from the VIRQ packet as the status byte. Use a mask byte of %00000001, defined as Vi.IFlag in the defs file. Use a flip byte value of 0. The following examples show how to set up both types of VIRQ calls. The first example is taken from an ACIA-type driver that has a physical interrupt found in a status register, but that cannot be accessed by the processor if used in the Multi-Pak. The second example is for a device with no physical interrupt handling, all interrupts are handled through the VIRQ.

* VIRQ Example #1 - Device Driver possessing real IRQ's

- * Copyright 1985,1986 by Microware Systems Corporation. Reproduced Under License

use defsfile

* actual mask byte for hardware interrupt
IRQReq set 0100000000 Interrupt Request

;
.

* offset to the actual hardware status register
Status equ 1

* VIRQ countdown value
VIRQCNT equ 1 do the VIRQ on every tick

;
.

* Static storage offsets

*

org V.SCF room for scf variables

;
.

VIRQBUF rmb 5 buffer for fake interrupt from clock

;
.

MEM equ . Total static storage requirement

* Module Header

mod MEND,NAM,DRIVR+OBJECT,REENT+1,ENT,MEM
fcb UPDAT.

fcb Edition Current Revision

* Driver entry jump table

ENT lbra INIT

lbra READ

lbra WRITE

lbra GETSTA

```
lbra PUTSTA
bra TRMNAT
```

- * Actual mask information for F\$IRQ call for the
- * hardware interrupt MASK fcb 0 no flip bits
- * fcb IRQReg Irq polling mask
- * fcb 10 (higher) priority

```
*****
```

- * Init
- * Initialize the device
- * Includes setting up the IRQ and VIRQ entries
- *

```
INIT
```

```
.
```

- * Install IRQ polling Table Entry first
- * Use the hardware status register and the hardware
- * mask

```
ldd V.PORT,U get port address in D
add #Status point to hardware status byte
leax MASK,PCR get the hardware interrupt mask
leay MIRQ,PCR address of interrupt service routine
OS9 F$IRQ Add to IRQ polling table
bcs INIT9 error - return it
```

- * Install VIRQ in Clock Module second
- *

```
leay VIRQBUF,U get the 5 byte VIRQ buffer pointer
lda #80 get reset flag for repeated VIRQ's
sta V1.Stat,y put it into buffer
ldd #VIRQCNT get count for number of ticks for the VIRQ
std V1.Rst,y put in initial reset value
ldx #1 put onto table
os9 F$VIRQ make the service request
bcs INIT9 Error - return it
```

```
.
```

```
INIT9 rts
```

```
READ
```

```
.
```

```
.
WRITE
GETSTA
PUTSTA
.
.

*****
* Subroutine TRMNAT
*   Terminate device, including removal from tables
*
TRMNAT
.
.
* remove from VIRQ table first
ldx #0 remove from VIRQ table
leay VIRQBUF,U get address
os9 F$VIRQ remove modem from VIRQ table
* next remove from IRQ table
ldx #0
OS9 F$IRQ remove modem from polling tbl
rts

*****
* MIRQ
*   process Interrupt
*
MDIRQ
.
.
< actual interrupt service routine >
.
rts

emod Module Crc
MEND equ *
```

* VIRQ Example #2 - Device Driver without hardware interrupts

```
*****
* STATIC STORAGE DEFINITION
*
```

•

.

.

VIRQBF rmb 5 buffer for VIRQ

DMEM equ .

• Module Header

•

mod DEND,DNAM,DRIVR+OBJECT,REENT+REV,DENT,DMEM

fcv UPDAT. mode byte

fcv 3 EDITION BYTE

• Driver entry table

DENT lbra INIT initialize

lbra READ

lbra WRITE

lbra GETSTAT get status

lbra SETSTAT set status

lbra TERM terminate

• Mask information packet for F\$IRQ call

• NOTE: uses the virtual interrupt flag, Vi.IFlag, for

• the maskbyte

•

DMSK fcb 0 no flip bits

fcv Vi.IFlag polling mask for VIRQ

fcv 10 priority

• INITIALIZE STORAGE AND CONTROLLER

• Includes setting up the IRQ and VIRQ table entries

•

INIT

• set up IRQ table entry first

• NOTE: uses the status register of the VIRQ buffer for

• the interrupt status register since no hardware status

• register is available

•

leay VIRQBF+Vi.Stat,U get address of status byte

```

tfr y,d put it into D reg
leay DIRQ,PCR get address of interrupt routine
leax DMSK,PCR get VIRQ mask info
os9 F$IRQ install onto table
bcs INIT9 exit on error

* now set up the VIRQ table entry
leay VIRQBF,U point to the 5-byte packet
lda #$80 get the reset flag to repeat VIRQ's
sta Vi.Stat,y save it in the buffer
ldd #VIRQCNT get the VIRQ counter value
std Vi.Rst,y save it in the reset area of buffer
ldx #1 code to install the VIRQ
os9 F$VIRQ install on the table
bcs INIT9 exit on error
.
.

INIT9 rts
.
.

READ
WRITE
GETSTAT
PUTSTAT
.
*****
* TERM - terminate the device and remove entries from
* tables
TERM
.
* remove from VIRQ table first
ldx #0 get zero to remove from table
leay VIRQBF,U get address of packet
os9 F$VIRQ
* then remove from IRQ table
ldx #0 get zero to remove from table
os9 F$IRQ
.
.
rts

```

```

*****

```

OS-9 Technical Reference

```
* DIRQ - interrupt service routine
*
* NOTE: The service routine must be sure to reset the
* status byte of the VIRQ packet so that the interrupt
* looks as if it is cleared.
```

```
*
DIRQ
.
.
lda VIRQBF+Vi.Stat,U get status byte
anda #$FF-Vi.IFlag mask off interrupt bit
sta VIRQBF+Vi.Stat,U put it back
.
.
rts
```

```
EMOD
DEND equ *
```

```
END
```

Memory Modules

In Chapter 2, you learned that OS-9 is based on the concept that memory is modular. This means that each program is considered to be an individually named *object*.

You also learned that each program loaded into memory must be in the module format. This format lets OS-9 manage the logical contents of memory, as well as the physical contents. Module types and formats are discussed in detail in this chapter.

Module Types

There are several types of modules. Each has a different use and function. These are the main requirements of a module:

- It cannot modify itself.
- It must be position-independent so that OS-9 can load or relocate it wherever space is available. In this respect, the module format is the OS-9 equivalent of *load records* used in older operating systems.

A module need not be a complete program or even 6809 machine language. It can contain BASIC09 I-code, constants, single sub-routines, and subroutine packages.

Module Format

Each module has three parts: a *module header*, a *module body*, and a *cyclic-redundancy-check value* (CRC value).

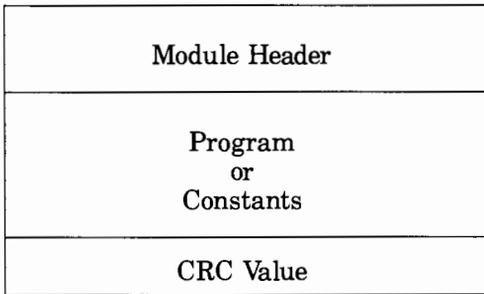


Figure 3.1

Module Header

At the beginning of the module (the lowest address) is the module header. Its form depends upon the module's use.

The header contains information about the module and its use. This information includes the following:

- Size
- Type (machine code, BASIC09 compiled code, and so on)
- Attributes (executable, re-entrant, and so on)
- Data storage memory requirements
- Execution starting address

Usually, you do not need to write routines to generate the modules and headers. All OS-9 programming languages automatically create modules and headers.

Module Body

The module body contains the program or constants. It usually is pure code. The module name string is included in this area. Figure 3.2 provides the offset values for calculating the location of a module's name. (See "Offset to Module Name".)

CRC Value

The last three bytes of the module are the Cyclic Redundancy Check (CRC) value. The CRC value is used to verify the integrity of a module.

When the system first loads the module into memory, it performs a 25-bit CRC over the entire module, from the first byte of the module header to the byte immediately before the CRC. The CRC polynomial used is \$800FE3.

As with the header, you usually don't need to write routines to generate the CRC value. Most OS-9 programs do this automatically.

Module Headers: Standard Information

The first nine bytes of all module headers are defined as follows:

Relative Address	Use
\$00,\$01	Sync bytes (\$87,\$CD)
\$02,\$03	Module size
\$04,\$05	Offset to module name
\$06	Module type/Language
\$07	Attributes/Revision level
\$08	Header check

Figure 3.2

Sync Bytes

The sync bytes specify the location of the module. (The first sync byte is the start of the module.) These two bytes are constant.

Module Size

The module size specifies the size of the module in bytes (includes CRC).

Offset to Module Name

The offset to module name specifies the address of the module name string relative to the start of the module. The name string can be located anywhere in the module. It consists of a string of ASCII characters with the most significant bit set on the last character.

Type/Language Byte

The type/language byte specifies the type and language of the module.

The four most significant bits of this byte indicate the type. Eight types are pre-defined. Some of these are for OS-9's internal use only. The type codes are given here (0 is not a legal type code):

Code	Module Type	Name
\$1x	Program module	Prgrm
\$2x	Subroutine module	Sbrtn
\$3x	Multi-module (for future use)	Multi
\$4x	Data module	Data
\$5x-\$Bx	User-definable module	
\$Cx	OS-9 system module	System
\$Dx	OS-9 file manager module	FlMgr
\$Ex	OS-9 device driver module	Drivr
\$Fx	OS-9 device descriptor module	Devic

Figure 3.3

The four least significant bits of Byte 6 indicate the language (denoted by x in the previous Figure). The language codes are given here:

Code	Language
\$x0	Data (non-executable)
\$x1	6809 object code
\$x2	BASIC09 I-code
\$x3	PASCAL P-code
\$x4-\$xF	Reserved for future use

Figure 3.4

By checking the language type, high-level language runtime systems can verify that a module is the correct type before attempting execution. BASIC09, for example, can run either I-code or 6809 machine language procedures arbitrarily by checking the language type code.

Attributes/Revision Level Byte

The attributes/revision level byte defines the attributes and revision level of the module.

The four most significant bits of this byte are reserved for module attributes. Currently, only Bit 7 is defined. When set, it indicates the module is re-entrant and, therefore, *shareable*.

The four least significant bits of this byte are a revision level in the range 0 to 15. If two or more modules have the same name, type, language, and so on, OS-9 keeps in the module directory only the module having the highest revision level. Therefore, you can replace or patch a ROM module, simply by loading a new, equivalent module that has a higher revision level.

Note: A previously linked module cannot be replaced until its link count goes to zero.

Header Check

The header check byte contains the one's complement of the Exclusive-OR of the previous eight bytes.

Module Headers: Type-Dependent Information

More information usually follows the first nine bytes of a module header. The layout and meaning vary, depending on the module type.

Module types \$Cx-\$Fx (system module, file manager module, device driver module, and device descriptor module) are used only by OS-9. Their formats are given later in the manual.

Module types \$1x through \$Bx have a general-purpose executable format. This format is often used in programs called by F\$Fork or F\$Chain. Here is the format used by these module types:

Executable Memory Module Format

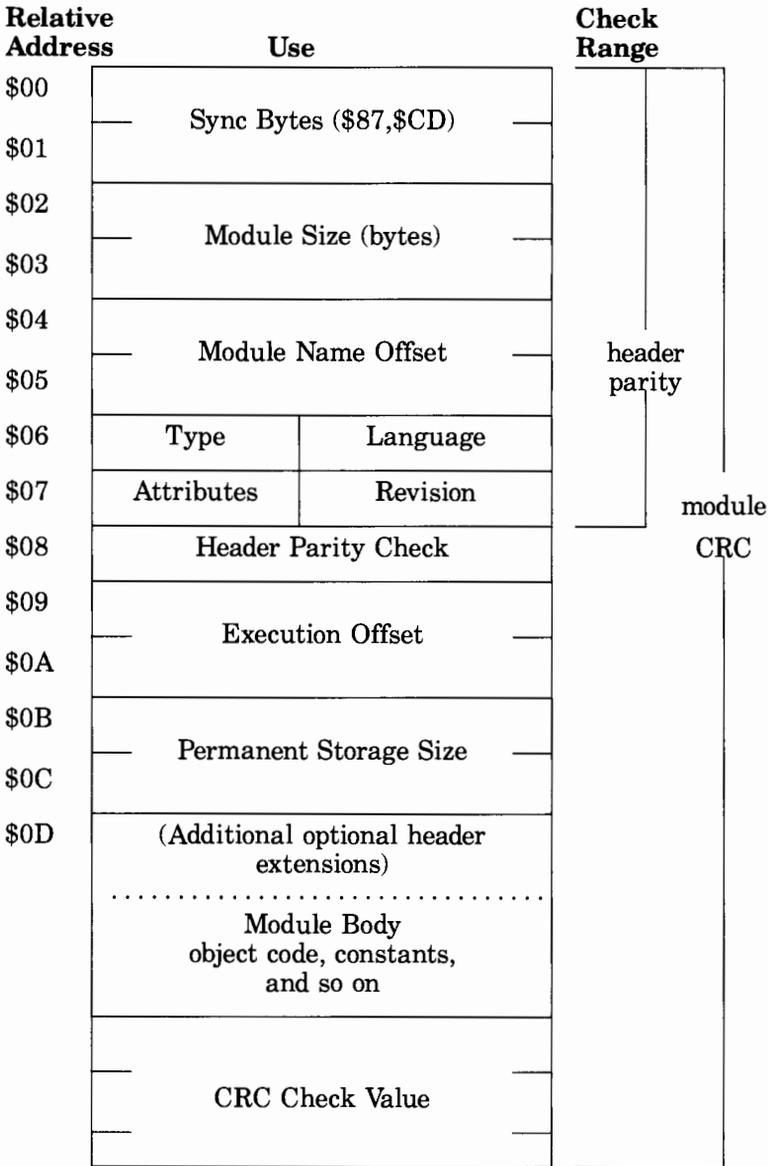


Figure 3.5

As you can see from the preceding chart, the executable memory has four extra bytes in its header. They are:

\$09,\$0A	Execution offset
\$0B,\$0C	Permanent storage size

Execution Offset. The program or subroutine's offset starting address, relative to the first byte of the sync code. A module that has multiple entry points (such as cold start and warm start) might have a branch table starting at this address.

Permanent Storage Size. The minimum number of bytes of data storage required to run. Fork and Chain use this number to allocate a process's data area.

If the module is not directly executed by a Fork or Chain system call (for instance a subroutine package), this entry is not used by OS-9. It is commonly used to specify the maximum stack size required by re-entrant subroutine modules. The calling program can check this value to determine if the subroutine has enough stack space.

When OS-9 starts after a single system reset, it searches the entire memory space for ROM modules. It finds them by looking for the module header sync code (\$87,\$CD).

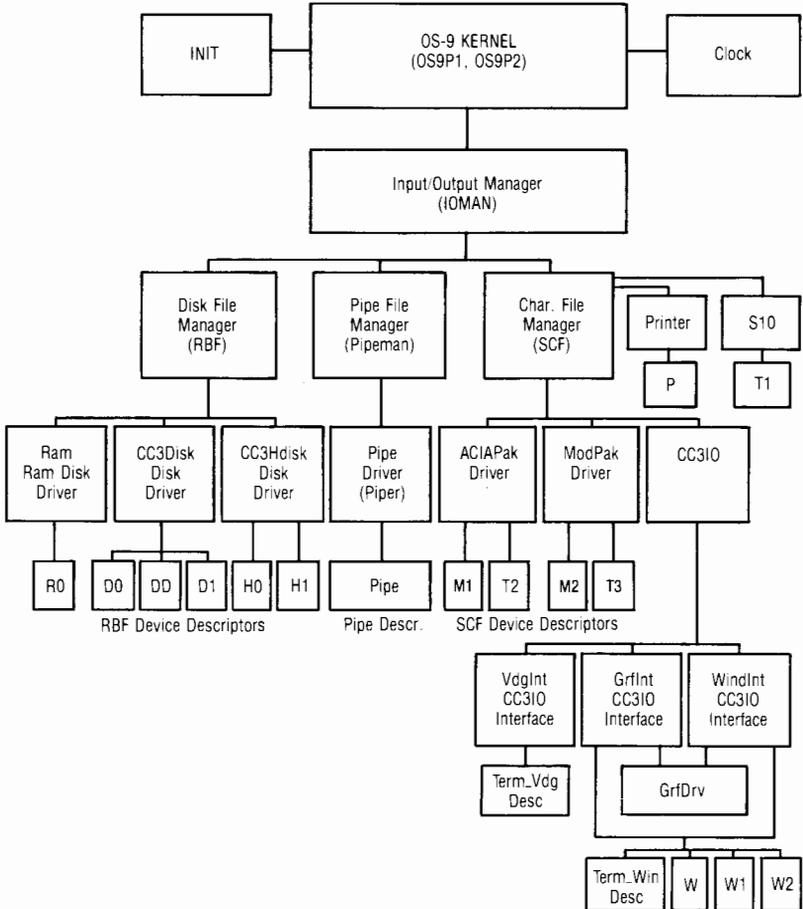
When OS-9 detects the header sync code, it checks to see if the header is correct. If it is, the system obtains the module size from the header and performs a 24-bit CRC over the entire module. If the CRC matches, OS-9 considers the module to be valid and enters it into the module directory. All ROM modules that are present in the system at startup are automatically included in the system module directory.

After the module search, OS-9 links to the component modules it found. This is the secret to OS-9's ability to adapt to almost any 6809 computer. It automatically locates its required and optional component modules and rebuilds the system each time it is started.

OS-9's Unified Input/Output System

Chapter 1 mentioned that OS-9 has a unified I/O system, consisting of all modules except those on the kernel level. This chapter discusses the I/O modules in detail.

I/O System Modules



OS-9 COMPONENT MODULE ORGANIZATION

The VDG Interface performs both interface and low level routines for VDG Color Computer 2 compatible modes and has limited support for high res screen allocation.

The GrfInt Interface provides the standard code interpretations and interface functions.

The WindInt Interface, available in the Multi-view package, contains all the functionality of GrfInt, along with additional support features. If you use WindInt, do not include GrfInt.

Both WindInt and GrfInt use the low-level driver GrfDrv to perform drawing on the bit-map screens.

Term_VDG uses CC3IO/VdgInt while Term_win and all window descriptors use CC3IO/(WindInt/GrfInt)/GrfDrv modules.

The I/O system provides system-wide, hardware-independent I/O services for user programs and OS-9 itself. All I/O system calls are received by the kernel and passed to the I/O manager for processing.

The I/O manager performs some processing, such as the allocation of data structures for the I/O path. Then, it calls the file managers and device drivers to do most of the work. Additional file manager, device driver, and device descriptor modules can be loaded into memory from files and used while the system is running.

The I/O Manager

The I/O manager provides the first level of service of I/O system calls. It routes data on I/O process paths to and from the appropriate file managers and device drivers.

The I/O Manager also maintains two important internal OS-9 data structures—the *device table* and the *path table*. Never modify the I/O manager.

When a path is opened, the I/O manager tries to link to a memory module that has the device name given or implied in the pathlist. This module is the *device descriptor*. It contains the names of the device driver and file manager for the device. The I/O manager saves the names so later system calls can be routed to these modules.

File Managers

OS-9 can have any number of *file manager modules*. Each of these modules processes the raw data stream to or from a class of device drivers that have similar operational characteristics. It removes as many unique characteristics as possible from I/O operations. Thus, it assures that similar devices conform to the OS-9 standard I/O and file structure.

The file manager also is responsible for mass storage allocation and directory processing, if these are applicable to the class of devices it serves.

File managers usually buffer the data stream and issue requests to the kernel for dynamic allocation of buffer memory. They can also monitor and process the data stream, for example, adding line-feed characters after carriage-return characters.

The file managers are re-entrant. The three standard OS-9 file managers are:

- Random block file manager: The RBF manager supports random-access, block-structured devices such as disk systems and bubble memories. (Chapter 5 discusses the RBF manager in detail.)
- Sequential Character File Manager: The SCF manager supports single-character-oriented devices, such as CRTs or hardcopy terminals, printers, and modems. (Chapter 6 discusses SCF in detail.)
- Pipe File Manager (PIPEMAN): The pipe manager supports interprocess communication via *pipes*.

File Manager Structure

Every file manager must have a branch table in exactly the following format. Routines that are not used by the file manager must branch to an error routine, that sets the carry and loads Register B with an appropriate error code before returning. Routines returning without error must ensure that the carry bit is clear.

- * All routines are entered with:
- * (Y) = Path Descriptor pointer
- * (U) = Caller's register stack pointer

```
EntryPt equ *  
  lbra Create  
  lbra Open  
  lbra MakDir  
  lbra ChgDir  
  lbra Delete  
  lbra Seek  
  lbra Read  
  lbra Write  
  lbra ReadLn  
  lbra WriteLn  
  lbra GetStat  
  lbra PutStat  
  lbra Close
```

Create, Open

Create and Open handle file creating and opening for devices. Typically, the process involves allocating any required buffers, initializing path descriptor variables, and establishing the path name. If the file manager controls multi-file devices (RBF), directory searching is performed to find or create the specified file.

Makdir

Makdir creates a directory file on multi-file devices. Makdir is neither preceded by a Create nor followed by a Close. File managers that are incapable of supporting directories need to return carry set with an appropriate error code in Register B.

ChgDir

On multi-file devices, ChgDir searches for a directory file. If ChgDir finds the directory, it saves the address of the directory (up to four bytes) in the caller's process descriptor. The descriptor is located at P\$DIO+2 (for a data directory) or P\$DIO+8 (for an execution directory).

In the case of the RBF manager, the address of the directory's file descriptor is saved. Open/Create begins searching in the current directory when the caller's pathlist does not begin with a slash (/). File managers that do not support directories should return the carry set and an appropriate error code in Register B.

Delete

Multi-file device managers handle file delete requests by initiating a directory search that is similar to Open. Once a device manager finds the file, it removes the file from the directory. Any media in use by the file are returned to unused status. In the case of the RBF manager, space is returned for system use and is marked as available in the free cluster bit map on the disk. File managers that do not support multi- file devices return an error.

Seek

File managers that support random access devices use Seek to position file pointers of an already open path to the byte specified. Typically, the positioning is a logical movement. No error is produced at the time of the seek if the position is beyond the current "end of file".

Normally, file managers that do not support random access ignore Seek. However, an SCF-type manager can use Seek to perform cursor positioning.

Read

Read returns the number of bytes requested to the user's data buffer. Make sure Read returns an EOF error if there is no data available. Read must be capable of copying pure binary data, and generally performs no editing on the data. Generally, the file manager calls the device driver to actually read the data into the buffer. Then, the file manager copies the data from the buffer into the user's data area to keep file managers device-independent.

Write

The Write request, like Read, must be capable of recording pure binary data without alteration. The routines for Read and Write are almost identical with the exception that Write uses the device driver's output routine instead of the input routine. The RBF manager and similar random access devices that use fixed-length records (sectors) must often pre-read a sector before writing it, unless they are writing the entire sector. In OS-9, writing past the end of file on a device expands the file with new data.

ReadLn

ReadLn differs from Read in two respects. First, ReadLn terminates when the first end-of-line (carriage return) is encountered. ReadLn performs any input editing that is appropriate for the device. In the case of SCF, editing involves handling functions such as backspace, line deletion, and the removal of the high-order bit from characters.

WriteLn

WriteLn is the counterpart of ReadLn. It calls the device driver to transfer data up to and including the first (if any) carriage return encountered. Appropriate output editing can also be performed. For example, SCF outputs a line feed, a carriage return character, and nulls (if appropriate for the device). It also pauses at the end of a screen page.

GetStat, PutStat

The GetStat (get status) and PutStat (put status) system calls are wildcard calls designed to provide a method of accessing features of a device (or file manager) that are not generally device independent. The file manager can perform specific functions such as setting the size of a file to a given value. Pass *unknown* status calls to the driver to provide further means of device independence. For example, a PutStat call to format a disk track might behave differently on different types of disk controllers.

Close

Close is responsible for ensuring that any output to a device is completed. (If necessary, Close writes out the last buffer.) It releases any buffer space allocated in an Open or Create. Close does not execute the device driver's terminate routine, but can do specific end-of-file processing if you want it to, such as writing end-of-file records on disks, or form feeds on printers.

Interfacing with Device Drivers

Strictly speaking, device drivers must conform to the general format presented in this manual. The I/O Manager is slightly different because it only uses the Init and Terminate entry points. Other entry points need only be compatible with the file manager for which the driver is written. For example, the Read entry point of an SCF driver is expected to return one byte from the device. The Read entry point of an RBF driver, on the other hand, expects Read to return an entire sector.

The following code is part of an SCF file manager. The code shows how a file manager might call a driver.

```
*****
* IOEXEC
*   Execute Device's Read/Write Routine
*
* Passed:  (A) = Output character (write)
*          (X) = Device Table entry ptr
*          (Y) = Path Descriptor pointer
*          (U) = Offset of routine (D$Read,
*              D$Write)
* Returns: (A) = Input char (read)
*          (B) = Error code, CC set if error
* Destroys B,CC
```

```
IOEXEC pshs a,x,y,u save registers
ldu V$STAT,x get static storage for driver
ldx V$DRIV,x get driver module address
ldd M$EXEC,x and offset of execution entries
addd 5,s offset by read/write
leax d,x absolute entry address
lda ,s+ restore char (for write)
jsr 0,x execute driver read/write
puls x,y,u,pc return (A)=char, (B)=error
```

```
emod Module CRC
Size equ * size of sequential file manager
```

Device Driver Modules

The device driver modules are subroutine packages that perform basic, low-level I/O transfers to or from a specific type of I/O device hardware controller. These modules are re-entrant. So, one copy of the module can concurrently run several devices that use identical I/O controllers.

Device driver modules use a standard module header, in which the module type is specified as code \$Ex (device driver). The execution offset address in the module header points to a branch table that has a minimum of six 3-byte entries.

Each entry is typically an LBRA to the corresponding subroutine. The file managers call specific routines in the device driver through this table, passing a pointer to a path descriptor and passing the hardware control register address in the 6809 registers. The branch table looks like this:

Code	Meaning
+\$00	Device initialization routine
+\$03	Read from device
+\$06	Write to device
+\$09	Get device status
+\$0C	Set device status
+\$0F	Device termination routine

(For a complete description of the parameters passed to these subroutines, see the "Device Driver Subroutines" sections in Chapters 5 and 6.)

Device Driver Module Format

Relative Address	Use	Check Range		
\$00	Sync Bytes (\$87,\$CD)	Header Parity		
\$01				
\$02	Module Size (bytes)			
\$03				
\$04	Module Name Offset			
\$05				
\$06	Type		Language	
\$07	Attributes		Revision	
\$08	Header Parity Check		Module CRC	
\$09	Execution Offset			
\$0A				
\$0B	Permanent Storage Size			
\$0C				
\$0D	Mode Byte			
	Module Body			
	CRC Check Value			

\$0D Mode Byte - (D S PE PW PR E W R)

OS-9 Interaction With Devices

Device drivers often must wait for hardware to complete a task or for a user to enter data. Such a wait situation occurs if an SCF device driver receives a Read but there is no data available, or if it receives a Write and no buffer space is available. OS-9 drivers that encounter this situation should suspend the current process (via F\$Sleep). In this way the driver allows other processes to continue using CPU time.

The most efficient way for a driver to awaken itself and resume processing data is by using interrupt requests (IRQs). It is possible for the driver to sleep for a number of system clock ticks and then check the device or buffer for a ready signal. The drawbacks to this technique are:

- It requires the system clock to always remain active.
- It might require a large number of ticks (perhaps 20) for the device to become ready. Such a case leaves you with a dilemma. If you make the program sleep for two ticks, the system wastes CPU time while checking for device ready. If the driver sleeps 20 ticks, it does not have a good response time.

An interrupt system allows the hardware to report to the CPU and the device drivers when the device is finished with an operation. Using interrupts to its advantage, a device driver can set up interrupt handling to occur when a character is sent or received or when a disk operation is complete. There is a built-in polling facility for pausing and awakening processes. Here is a technique for handling interrupts in a device driver:

1. Use the Init routine to place the driver interrupt service call (IRQSVC) routine in the IRQ polling sequence via an F\$IRQ system call:

```
ldd V.Port,u get address to poll
leax IRQPOLL,pcr point to IRQ packet
leay IRQSVC,pcr point to IRQ routine
OS9 F$IRQ add dev to poll sequence
bcs Error abnormal exit if error
```

2. Ensure that driver programs waiting for their hardware, call the sleep routine. The sleep routine copies V.Busy to V.Wake. Then, it goes to sleep for some period of time.

3. When the driver program wakes up, have it check to see whether it was awakened by an interrupt or by a signal sent from some other process.

Usually, the driver performs this check by reading the V.Wake storage byte. The V.Busy byte is maintained by the file manager to be used as the process ID of the process using the driver. When V.Busy is copied into V.Wake, then V.Wake becomes a flag byte and an information byte. A non-zero Wake byte indicates that there is a process awaiting an interrupt. The value in the Wake byte indicates the process to be awakened by sending a wakeup signal as shown in the following code:

```
        lda V.Busy,u           get proc ID
        sta V.Wake,u          arrange for wakeup
        andcc #^IntMasks      prep for interrupts
Sleep50 ldx #0                 or any other tick time
                                (if signal test)
        OS9 F$Sleep           await an IRQ
        ldx D.Proc            get proc desc ptr if
                                signal test
        ldb P$Signal,x        is signal present?
                                (if signal test)
        bne SigTest           bra if so if signal
                                test
        tst V.Wake,u          IRQ occur?
        bne Sleep50          bra if not
```

Note that the code labeled "if signal test" is only necessary if the driver wishes to return to the caller if a signal is sent without waiting for the device to finish. Also note that IRQs and FIRQs must be masked between the time a command is given to the device and the moving of V.Busy and V.Wake. If they are not masked, it is possible for the device IRQ to occur and the IRQSVC routine to become confused as to whether it is sending a wakeup signal or not.

4. When the device issues an interrupt, OS-9 calls the routine at the address given in F\$IRQ with the interrupts masked. Make the routine as short as possible, and have it return with an RTS instruction. IRQSVC can verify that an interrupt has occurred for the device. It needs to clear the interrupt to retrieve any data in the device. Then the V.Wake byte communicates with the main driver module. If V.Wake is non-zero, clear it to indicate a true device interrupt and use its contents as the process ID for an F\$Send system call. The F\$Send call sends a wakeup signal to the process. Here is an example:

```
    ldx V.Port,u get device address
    tst ?? is it real interrupt from device?
    bne IRQSVC90 bra to error if not
    lda Data,x get data from device
    sta 0,y
    lda V.Wake,u
    beq IRQSVC80 bra if none
    clr V.Wake,u clear it as flag to main
    routine
    ldb #S$Wake,u get wakeup signal
    OS9 F$Send send signal to driver
IRQSVC80 clrb clear carry bit (all is well)
         rts
IRQSVC90 comb set carry bit (is an IRQ call)
         rts
```

Suspend State (Level Two only)

The Suspend State allows the elimination of the F\$Send system call during interrupt handling. Because the process is already in the active queue, it need not be moved from one queue to another. The device driver IRQSVC routine can now wake up the suspended main driver by clearing the process status byte suspend bit in the process state. Following are sample routines for the Sleep and IRQSVC calls:

```
    lda D.Proc get process ptr
    sta V.Wake,u prep for re-awakening

    enable device to IRQ, give command, etc.

    bra Cmd50 enter suspend loop

Cmd30 ldx D.Proc get ptr to process desc
```

```
    lda P$State,x get state flag
    ora #Suspend put proc in suspend state
    sta P$State,x save it in proc desc
    andcc #^IntMasks unmask interrupts
    ldx #1 give up time slice
    OS9 F$Sleep suspend (in active queue)
Cmd50 orcc #IntMasks mask interrupts while
changing state
    ldx D.Proc get proc desc addr (if signal
test)
    lda P$Signal,x get signal (if signal test)
    beq SigProc bra if signal to be handled
    lda V.Wake,u true interrupt?
    bne Cmd30 bra if not
    andcc #^IntMasks assure interrupts unmasked
```

Note that D.Proc is a pointer to the process descriptor of the current process. Process descriptors are always allocated on 256-byte page boundaries. Thus, having the high order byte of the address is adequate to locate the descriptor. D.Proc is put in V.Wake as a dual value. In one instance, it is a flag byte indicating that a process is indeed suspended. In the other instance, it is a pointer to the process descriptor which enables the IRQSVC routine to clear the suspend bit. It is necessary to have the interrupts masked from the time the device is enabled until the suspend bit has been set. Making the interrupts ensure that the IRQSVC routine does not think it has cleared the suspend bit before it is even set. If this happens, when the bit is set the process might go into permanent suspension. The IRQSVC routine sample follows:

```
    ldy V.Port,u get dev addr
    tst V.Wake,u is process awaiting
    IRQ?
    beq IRQSVCKER no exit

    clear device interrupt
    exit if IRQ not from this device

    lda V.Wake,u get process ptr
    clrb
    stb V.Wake,u clear proc waiting flag
    tfr d,x get process descriptor ptr
    lda P$State,x get state flag
    anda # Suspend clear suspend state
    sta P$State,x save it
```

```

        clr b clear carry bit
        rts
    IRQSV CER comb set carry bit
        rts
    
```

Device Descriptor Modules

Device descriptor modules are small, non-executable modules. Each one provides information that associates a specific I/O device with its logical name, hardware controller address(es), device driver, file manager name, and initialization parameters.

Unlike the device drivers and file managers, which operate on classes of devices, each device descriptor tailors its functions to a specific device. Each device must have a device descriptor.

Device descriptor modules use a standard module header, in which the module type is specified as code \$Fx (device descriptor). The name of the module is the name by which the system and user know the device (the device name given in pathlists).

The rest of the device descriptor header consists of the information in the following chart:

Relative Address(es)	Use
\$09,\$0A	The relative address of the file manager name string address
\$0B,\$0C	The relative address of the device driver name string
\$0D	Mode/Capabilities: D S PE PW PR E W R (directory, single user, public execute, public write, public read, execute, write, read)
\$0E,\$0F,\$10	The absolute physical (24-bit) address of the device controller
\$11	The number of bytes (<i>n</i> bytes) in the initialization table
\$12,\$12 + <i>n</i>	Initialization table

When OS-9 opens a path to the device, the system copies the initialization table into the option section (PD.OPT) of the path descriptor. (See "Path Descriptors" in this chapter.)

The values in this table can be used to define the operating parameters that are alterable by the Get Status and Set Status system calls (I\$GetStt and I\$SetStt). For example, parameters that are used when initializing terminals define which control characters are to be used for functions such as backspace and delete.

The initialization table can be a maximum of 32 bytes long. If the table is fewer than 32 bytes long, OS-9 sets the remaining values in the path descriptor to 0.

You might wish to add devices to your system. If a similar device driver already exists, all you need to do is add the new hardware and load another device descriptor. Device descriptors can be in the boot module or they can be loaded into RAM from mass-storage files while the system is running.

The following diagram illustrates the device descriptor format:

Device Descriptor Format

Relative Address	Use	Check Range	
\$00	Sync Bytes (\$87,\$CD)	header parity	
\$01			
\$02	Module Size (bytes)		
\$03			
\$04	Offset to Module Name		
\$05			
\$06	F\$ (Type)		\$l (Lang)
\$07	Attributes		Revision
\$08	Header Parity Check		
\$09	Offset to File Manager Name String		module CRC
\$0A			
\$0B	Offset to Device Driver Name String		
\$0D			
\$0E	Mode Byte		
\$0F	Device Controller Absolute Physical Addr. (24 bit)		
\$10			
\$11	Initialization Table Size		
\$12,\$12 + n	(Initialization Table)		
	(Name Strings, and so on)		
	CRC Check Value		

Path Descriptors

Every open path is represented by a data structure called a *path descriptor* (PD). The PD contains the information the file managers and device drivers require to perform I/O functions.

PDs are 64 bytes long and are dynamically allocated and deallocated by the I/O manager as paths are opened and closed.

They are internal data structures, that are not normally referenced from user or applications programs. The description of PDs is presented here mainly for those programmers who need to write custom file managers, device drivers, or other extensions to OS-9.

PDs have three sections. The first section, which is ten bytes long, is the same for all file managers and device drivers. The information in the first section is shown in the following chart.

Path Descriptor: Standard Information

Name	Relative Address	Size (Bytes)	Use
PD.PD	\$00	1	Path number
PD.MOD	\$01	1	Access mode: 1 = read, 2 = write, 3 = update
PD.CNT	\$02	1	Number of open paths using this PD
PD.DEV	\$03	2	Address of the associated device table entry
PD.CPR	\$05	1	Current process ID
PD.RGS	\$06	2	Address of the caller's register stack
PD.BUF	\$08	2	Address of the 256-byte data buffer (if used)
PD.FST	\$0A	22	Defined by the file manager
PD.OPT	\$20	32	Reserved for the Getstat/Setstat options

PD.FST is 22-byte storage reserved for and defined by each type of file manager for file pointers, permanent variables, and so on.

PD.OPT is a 32-byte option area used for file or device operating parameters that are dynamically alterable. When the path is opened, the I/O manager initializes these variables by copying the initialization table that is in the device descriptor module. User programs can change the values later, using the Get Status and Set Status system calls.

PD.FST and **PD.OPT** are defined for the file manager in the assembly-language equate file (SCFDefs for the SCF manager or RBFDefs for the RBF manager).

Random Block File Manager

The random block file manager (RBF manager) supports *disk storage*. It is a re-entrant subroutine package called by the I/O manager for I/O system calls to random-access devices. It maintains the logical and physical file structures.

During normal operation, the RBF manager requests allocation and deallocation of 256-byte data buffers. Usually, one buffer is required for each open file. When physical I/O functions are necessary, the RBF manager directly calls the subroutines in the associated device drivers. All data transfers are performed using 256-byte data blocks (pages).

The RBF manager does not deal directly with physical addresses such as tracks and cylinders. Instead, it passes to the device drivers address parameters, using a standard address called a *logical sector number*, or *LSN*. LSNs are integers from 0 to $n-1$, where n is the maximum number of sectors on the media. The driver translates the logical sector number to actual cylinder/track/sector values.

Because the RBF manager supports many devices that have different performance and storage capacities, it is highly parameter-driven. The physical parameters it uses are stored on the media itself.

On disk systems, the parameters are written on the first few sectors of Track 0. The device drivers also use the information, particularly the physical parameters stored on Sector 0. These parameters are written by the FORMAT program that initializes and tests the disk.

Logical and Physical Disk Organization

All disks used by OS-9 store basic identification, file structure, and storage allocation information on these first few sectors.

LSN 0 is the *identification sector*. LSN 1 is the *disk allocation map sector*. LSN 2 marks the beginning of the disk's ROOT directory. The following section tells more about LSN 0 and LSN 1.

Identification Sector (LSN 0)

LSN 0 contains a description of the physical and logical characteristics of the disk. These characteristics are set by the **FORMAT** command program when the disk is initialized.

The following table gives the OS-9 mnemonic name, byte address, size, and description of each value stored in this LSN 0.

Name	Relative Address	Size (Bytes)	Use
DD.TOT	\$00	3	Number of sectors on disk
DD.TKS	\$03	1	Track size (in sectors)
DD.MAP	\$04	2	Number of bytes in the allocation bit map
DD.BIT	\$06	2	Number of sectors per cluster
DD.DIR	\$08	3	Starting sector of the ROOT directory
DD.OWN	\$0B	2	Owner's user number
DD.ATT	\$0D	1	Disk attributes
DD.DSK	\$0E	2	Disk identification (for internal use)
DD.FMT	\$10	1	Disk format, density, number of sides
DD.SPT	\$11	2	Number of sectors per track
DD.RES	\$13	2	Reserved for future use
DD.BT	\$15	3	Starting sector of the bootstrap file
DD.BSZ	\$18	2	Size of the bootstrap file (in bytes)
DD.DAT	\$1A	5	Time of creation (Y:M:D:H:M)
DD.NAM	\$1F	32	Volume name in which the last character has the most significant bit set
DD.OPT	\$3F		Path descriptor options

Disk Allocation Map Sector (LSN 1)

LSN 1 contains the *disk allocation map*, which is created by FORMAT. This map shows which sectors are allocated to the files and which are free for future use.

Each bit in the allocation map represents a sector or cluster of sectors on the disk. If the bit is set, the sector is considered to be in use, defective, or non-existent. If the bit is cleared, the corresponding cluster is available. The allocation map usually starts at LSN1. The number of sectors it requires varies according to how many bits are needed for the map. DD.MAP specifies the actual number of bytes used in the map.

Multiple sector allocation maps allow the number of sectors/cluster to be as small as possible for high volume media.

The FORMAT utility bases the size of the allocation map on the size and number of sectors per cluster.

The DD.MAP value in LSN 0 specifies the number of bytes (in LSN 1) that are used in the map.

Each bit on the disk allocation map corresponds to one sector cluster on the disk. The DD.BIT value in LSN 0 specifies the number of sectors per cluster. The number is an integral power of 2 (1, 2, 4, 8, 16, and so on).

If a cluster is available, the corresponding bit is cleared. If it is allocated, non-existent, or physically defective, the corresponding bit is set.

ROOT Directory

This file is the parent directory of all other files and directories on the disk. It is the directory accessed using the physical device name (such as /D1). Usually, it immediately follows the Allocation Map. The location of the ROOT directory file descriptor is specified in DD.DIR. The ROOT directory contains an entry for each file that resides in the directory, including other directories.

File Descriptor Sector

The first sector of every file is the *file descriptor*. It contains the logical and physical description of the file.

The following table describes the contents of the file descriptor.

Name	Relative Address	Size (Bytes)	Use
FD.ATT	\$00	1	File attributes: D S PE PW PR E W R (see next chart)
FD.OWN	\$01	2	Owner's user ID
FD.DAT	\$03	5	Date last modified: (Y M D H M)
FD.LNK	\$08	1	Link count
FD.SIZ	\$09	4	File size (number of bytes)
FD.CREAT	\$0D	3	Date created (Y M D)
FD.SEG	\$10	240	Segment list (see next chart)

FD.ATT. (The attribute byte) contains the file permission bits. When set the bits indicate the following:

- Bit 7 Directory
- Bit 6 Single user
- Bit 5 Public execute
- Bit 4 Public write
- Bit 3 Public read
- Bit 2 Execute
- Bit 1 Write
- Bit 0 Read

FD.SEG (the segment list) consists of a maximum of 48 5-byte entries that have the size and address of each file block in logical order. Each entry has the block's 3-byte LSN and 2-byte size (in sectors). The entry following the last segment is zero.

After creation, a file has no data segments allocated to it until the first write. (Write operations past the current end-of-file cause sectors to be added to the file. The first write is always past the end-of-file.)

If the file has no segments, it is given an initial segment. Usually, this segment has the number of sectors specified by the minimum allocation entry in the device descriptor. If, however, the number of sectors requested is more than the minimum, the initial segment has the requested number.

Later expansions of the file usually are also made in minimum allocation increments. Whenever possible, OS-9 expands the last segment, instead of adding a segment. When the file is closed, OS-9 truncates unused sectors in the last segment.

OS-9 tries to minimize the number of storage segments used in a file. In fact, many files have only one segment. In such cases, no extra Read operations are needed to randomly access any byte in the file.

If a file is repeatedly closed, opened, and expanded, it can become fragmented so that it has many segments. You can avoid this fragmentation by writing a byte at the highest address you want to be used on a file. Do this before writing any other data.

Directories

Disk directories are files that have the D attribute set. A directory contains an integral number of entries, each of which can hold the name and LSN of a file or another directory.

Each directory entry contains 29 bytes for the filename, followed by the three bytes for the LSN of the file's descriptor sector. The filename is left-justified in the field, with the most significant bit of the last character set. Unused entries have a zero byte in the first filename character position.

Every disk has a master directory called the ROOT directory. The DD.DIR value in LSN 0 (identification sector) specifies the starting sector of the ROOT directory.

The RBF Manager Definitions of the Path Descriptor

As stated earlier in this chapter, the PD.FST section of the path descriptor is reserved for and defined by the file manager. The following table describes the use of this section by the RBF manager. For your convenience, it also includes the other sections of the PD.

Name	Relative Address	Size (Bytes)	Use
Universal Section (Same for all file managers and device drivers)			
PD.PD	\$00	1	Path number
PD.MOD	\$01	1	Access mode 1 = read, 2 = write, 3 = update
PD.CNT	\$02	1	Number of open images (paths using this PD)
PD.DEV	\$03	2	Address of the associated device table entry
PD.CPR	\$05	1	Current process ID
PD.RGS	\$06	2	Address of the caller's 6809 register stack
PD.BUF	\$08	2	Address of the 256-byte data buffer (if used)

Name	Relative Address	Size (Bytes)	Use
The RBF manager Path Descriptor Definitions (PD.FST Section)			
PD.SMF	\$0A	1	State flag: Bit 0 = current buffer is altered Bit 1 = current sector is in the buffer Bit 2 = descriptor sector is in the buffer
PD.CP	\$0B	4	Current logical file position (byte address)
PD.SIZ	\$0F	4	File size
PD.SBL	\$13	3	Segment beginning logical sector number (LSN)
PD.SBP	\$16	3	Segment beginning physical sector number (PSN)

Name	Relative Address	Size (Bytes)	Use
PD.SSZ	\$19	3	Segment size
PD.DSK	\$1C	2	Disk ID (for internal use only)
PD.DTB	\$1E	2	Address of drive table

Name	Relative Address	Size (Bytes)	Use
-------------	-------------------------	---------------------	------------

The RBF manager Option Section Definitions (PD.OPT Section)

(Copied from the device descriptor)

PD.DTP	\$20	1	Device class: 0 = SCF 1 = RBF 2 = PIPE 3 = SBF
PD.DRV	\$21	1	Drive number (0..n)
PD.STP	\$22	1	Step rate
PD.TYP	\$23	1	Device type
PD.DNS	\$24	1	Density capability
PD.CYL	\$25	2	Number of cylinders (tracks)
PD.SID	\$27	1	Number of sides (surfaces)
PD.VFY	\$28	1	0 = verify disk writes
PD.SCT	\$29	2	Default number of sectors per track
PD.T0S	\$2B	2	Default number of sectors per track (Track 0)
PD.ILV	\$2D	1	Sector interleave factor
PD.SAS	\$2E	1	Segment allocation size
PD.TFM	\$2F	1	DMA transfer mode
PD.EXTEN	\$30	2	Path extension for record locking
PD.STOFF	\$32	1	Sector/track offsets

Name	Relative Address	Size (Bytes)	Use
(Not copied from the device descriptor):			
PD.ATT	\$33	1	File attributes (D S PE PW PR E W R)
PD.FD	\$34	3	File descriptor PSN
PD.DFD	\$37	3	Directory file descriptor PSN
PD.DCP	\$3A	4	File's directory entry pointer
PS.DVT	\$3E	2	Address of the device table entry

Any values not determined by this table default to zero.

RBF-Type Device Descriptor Modules

This section describes the use of the initialization table contained in the device descriptor modules for RBF-type devices. The following values are those the I/O manager copies from the device descriptor to the path descriptor.

Name	Relative Address	Size (Bytes)	Use
	\$0-\$11		Standard device descriptor module header
IT.DTP	\$12	1	Device type: 0 = SCF 1 = RBF 2 = PIPE 3 = SBF
IT.DRV	\$13	1	Drive number
IT.STP	\$14	1	Step rate
IT.TYP	\$15	1	Device type (see RBF path descriptor)
IT.DNS	\$16	1	Media density: Always 1 (double) (see following information)
IT.CYL	\$17	2	Number of cylinders (tracks)
IT.SID	\$19	1	Number of sides
IT.VFY	\$1A	1	0 = Verify disk writes 1 = no verify
IT.SCT	\$1B	2	Default number of sectors per track
IT.T0S	\$1D	2	Default number of sectors per track (Track 0)
IT.ILV	\$1F	1	Sector interleave factor
IT.SAS	\$20	1	Minimum size of segment allocation (number of sectors to be allocated at one time)

IT.DRV is used to associate a 1-byte integer with each drive that a controller handles. Number the drives for each controller as 0 to $n-1$, where n is the maximum number of drives the controller can handle.

IT.TYP specifies the device type (all types).

- Bit 0 — 0 = 5-inch floppy diskette
- Bit 5 — 0 = Non-Color Computer format
1 = Color Computer format
- Bit 6 — 0 = Standard OS-9 format
1 = Non-standard format
- Bit 7 — 0 = Floppy diskette
1 = Hard disk

IT.DNS specifies the density capabilities (floppy diskette only).

- Bit 0 — 0 = Single-bit density (FM)
1 = Double-bit density (MFM)
- Bit 1 — 0 = Single-track density (5-inch, 48 tracks per inch)
1 = Double-track density (5-inch, 96 tracks per inch)

IT.SAS specifies the minimum number of sectors allowed at one time.

RBF Record Locking

Record locking is a general term that refers to methods designed to preserve the integrity of files that can be accessed by more than one user or process. The OS-9 implementation of record locking is designed to be as invisible as possible. This means that existing programs do not have to be rewritten to take advantage of record locking facilities. You can usually write new programs without special concern for multi-user activity.

Record locking involves detecting and preventing conflicts during record access. Whenever a process modifies a record, the system locks out other procedures from accessing the file. It defers access to other procedures until it is safe for them to write to the record. The system does not lock records during reads; so, multiple processes can read the record at the same time.

Record Locking and Unlocking

To detect conflicts, OS-9 must recognize when a record is being updated. The RBF manager provides true record locking on a byte basis. A typical record update sequence is:

```
OS9 I$Read      program reads record
                 RECORD IS LOCKED
.
.
.               program updates record
.
OS9 I$Seek      reposition to record
OS9 I$Write     record is rewritten
                 RECORD IS RELEASED
```

When a file is opened in update mode, any read causes locking of the record being accessed. This happens because the RBF manager cannot determine in advance if the record is to be updated. The record stays locked out until the next read, write, or close.

However, when a file is opened in the read or execute modes, the system does not lock accessed records because the records cannot be updated in these two modes.

A subtle but important problem exists for programs that interrogate a data base and occasionally update its data. If you neglect to release a record after accessing it, the record might be locked up indefinitely. This problem is characteristic of record locking systems and you can avoid it with careful programming.

Only one portion of a file can be locked out at a time. If an application requires more than one record to be locked out, open multiple paths to the same file and lock the record accessed by each path. RBF notices that the same process owns both paths and keeps them from locking each other out.

Non-Shareable Files

Sometimes (although rarely), you must create a file that can never be accessed by more than one user at a time. To lock the file, you set the single-user (s) bit in the file's attribute byte. You can do this by using the proper option when the file is created, or later using the OS-9 ATTR command. Once the single-user bit is set, only one user can open the file at a time. If other users attempt to open the file, Error 253 is returned. Note however, that non-shareable means only one path can be opened to a file at one time. Do not allow two processes to concurrently access a non-shareable file through the same path.

More commonly, you need to declare a file as single-user only during the execution of a specific program. You can do this by opening the file with the single-user bit set. For example, suppose a process is sorting a file. With the file's single-user bit set, OS-9 treats the file exactly as though it had a single-user attribute. If another process attempts to open the file, OS-9 returns Error 253.

You can duplicate non-shareable paths by using the I\$Dup system call. This means that it can be inherited, and therefore accessible to more than one process at a time. Single-user means that the file can be opened only once.

End-of-File Lock

A special case of record locking occurs when a user reads or writes data at the end of a file, creating an *EOF Lock*. An EOF Lock keeps the end of the file locked out until a process performs a READ or WRITE that is not at the end of the file. It prevents problems that might otherwise occur when two users want to simultaneously extend a file. The EOF Lock is the only case in which a WRITE call automatically causes portions of a file to be locked out. An interesting and useful side effect of the EOF Lock function occurs if a program creates a file for sequential output. As soon as the program creates the file, EOF Lock is set and no other process can *pass* the writer in processing the file. For example, if an assembler redirects a listing to a disk file, and a spooler utility tries to print a line from the file before it is written, record locking makes the spooler wait and stay at least one step behind the assembler.

Deadlock Detection

A *deadly embrace*, or deadlock, typically occurs when two processes attempt to gain control of two or more disk areas at the same time. If each process gets one area (locking out the other process), both processes become permanently stuck. Each waits for a segment that can never become free. This situation is not restricted to any particular record locking scheme or operating system.

When a deadly embrace occurs, RBF returns a deadlock error (Error 254) to the process that caused OS-9 to detect the deadlock. To avoid deadlocks, make sure that processes always access records of shared files in the same sequence.

When a deadlock error occurs, it is not sufficient for a program to retry the operation that caused the error. If all processes use this strategy, none can ever succeed. For any process to proceed, at least one must cancel operation to release its control over a requesting segment.

RBF-Type Device Driver Modules

An RBF-type device driver module contains a package of subroutines that perform sector-oriented I/O to or from a specific hardware controller. Such a module is usually re-entrant. Because of this, one copy of one device driver module can simultaneously run several devices that use identical I/O controllers.

The I/O manager allocates a permanent memory area for each device driver. The size of the memory area is given in the device driver module header. The I/O manager and the RBF manager use some of this area. The device driver can use the rest in any manner. This area is used as follows:

The RBF Device Memory Area Definitions

Name	Relative Address	Size (Bytes)	Use
V.PAGE	\$00	1	Port extended address bits A20-A16
V.PORT	\$01	2	Device base address (defined by the I/O manager)

Name	Relative Address	Size (Bytes)	Use
V.LPRC	\$03	1	ID of the last active process (not used by RBF device drivers)
V.BUSY	\$04	1	ID of the current process using driver (defined by RBF) 0 = no current process
V.WAKE	\$05	1	ID of the process waiting for I/O completion (defined by the device driver)
V.USER	\$06	0	Beginning of file manager specific storage
V.NDRV	\$06	1	Maximum number of drives the controller can use (defined by the device driver)
	\$07	8	Reserved
DRVBEG	\$0F	0	Beginning of the drive tables
TABLES	\$0F	DRVMEN*N	Space for number of tables reserved (<i>n</i>)
FREE		0	Beginning of space available for driver

These values are defined in files in the DEFS directory on the Development Package disk.

TABLES. This area contains one table for each drive that the controller handles. (The RBF manager assumes that there are as many tables as indicated by V.NDRV.) Some time after the driver Init routine is called, the RBF manager issues a request for the driver to read LSN 0 from a drive table by copying the first part of LSN 0 (up to DD.SIZ) into the table. Following is the format of each drive table:

Name	Relative Address	Size (Bytes)	Use
DD.TOT	\$00	3	Number of sectors.
DD.TKS	\$03	1	Track size (in sectors).
DD.MAP	\$04	2	Number of bytes in the allocation bit map.
DD.BIT	\$06	2	Number of sectors per bit (cluster size).
DD.DIR	\$08	3	Address (LSN) of the ROOT directory.
DD.OWN	\$0B	2	Owner's user number.
DD.ATT	\$0D	1	Disk access attributes (D S PE PW PR E W R).
DD.DSK	\$0E	2	Disk ID (a pseudo-random number used to detect diskette swaps).
DD.FMT	\$10	1	Media format.
DD.SPT	\$11	2	Number of sectors per track. (Track 0 can use a different value specified by IT.TOS in the device descriptor.)
DD.RES	\$13	2	Reserved for future use.
DD.SIZ	\$15	0	Minimum size of device descriptor.
V.TRAK	\$15	2	Number of the current track (the track that the head is on, and the track updated by the driver).
V.BMB	\$17	1	Bit-map use flag: 0 = Bit map is not in use. (Disk driver routines must not alter V.BMB.)
V.FILEHD	\$18	2	Open file list for this drive.

Name	Relative Address	Size (Bytes)	Use
V.DISKID	\$1A	2	Disk ID.
V.BMAPSZ	\$1C	1	Size of bitmap.
V.MAPSCT	\$1D	1	Lowest reasonable bitmap sector.
V.RESBIT	\$1E	1	Reserved bitmap sector.
V.SCTKOF	\$1F	1	Sector/track byte.
V.SCOFST	\$20	1	Sector offset split from V.SCTKOF.
V.TKOFST	\$21	1	Track offset split from V.SCTKOF.
RESERVED	\$22	4	Reserved for future use.
DRVMEN ^A	\$26		Size of each drive table.

The format attributes (DD.FMT) are these:

Bit B0 = Number of sides
0 = Single-sided
1 = Double-sided

Bit B1 = Density
0 = Single-density
1 = Double-density

Bit B2 = Track density
0 = Single (48 tracks per inch)
1 = Double (96 tracks per inch)

RBF Device Driver Subroutines

Like all device driver modules, RBF device drivers use a standard executable memory module format.

The execution offset address in the module header points to a branch table that has six 3-byte entries. Each entry is typically a long branch (LBRA) to the corresponding subroutine. The branch table is defined as follows:

ENTRY	LBRA	INIT	Initialize drive
	LBRA	READ	Read sector
	LBRA	WRITE	Write sector
	LBRA	GETSTA	Get status
	LBRA	SETSTA	Set status
	LBRA	TERM	Terminate device

Ensure that each subroutine exists with the C bit of the condition code register cleared if no error occurred. If an error occurs, set the C bit and return an appropriate error code Register B.

The rest of this chapter describes the RBF device driver subroutines and their entry and exit conditions.

Init Initializes a device and the device's memory area.

Entry Conditions:

- Y = *address of the device descriptor*
- U = *address of the device memory area*

Exit Conditions:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- If you want OS-9 to verify disk writes, use the Request Memory system call (F\$SRqMem) to allocate a 256-byte buffer area in which a sector can be read back and verified after a write.
- You must initialize the device memory area. For floppy diskette controllers, initialization typically consists of:
 1. Initializing V.NDRV to the number of drives with which the controller works
 2. Initializing DD.TOT (in the drive table) to a non-zero value so that Sector 0 can be read or written
 3. Initializing V.TRAK to \$FF so that the first seek finds Track 0
 4. Placing the IRQ service routine on the IRQ polling list, using the Set IRQ system call (F\$IRQ)
 5. Initializing the device control registers (enabling interrupts if necessary)
- Prior to being called, the device memory area is cleared (set to zero), except for V.PAGE and V.PORT. (These areas contain the 24-bit device address.) Ensure the driver initializes each drive table appropriately for the type of diskette that the driver expects to be used on the corresponding drive.

Read Reads a 256-byte sector from a disk and places it in a 256-byte sector buffer.

Entry Conditions:

B = *MSB of the disk's LSN*
X = *LSB of the disk's LSN*
Y = *address of the path descriptor*
U = *address of the device memory area*

Exit Conditions:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- The following is a typical routine for using Read:
 1. Get the sector buffer address from PD.BUF in the path descriptor.
 2. Get the drive number from PD.DRV in the path descriptor.
 3. Compute the physical disk address from the logical sector number.
 4. Initiate the Read operation.
 5. Copy V.BUSY to V.WAKE. The driver goes to sleep and waits for the I/O to complete. (The IRQ service routine is responsible for sending a wakeup signal.) After awakening, the driver tests V.WAKE to see if it is clear. If it isn't clear, the driver goes back to sleep.
- Whenever you read LSN 0, you must copy the first part of this sector into the proper drive table. (Get the drive number from PD.DRV in the path descriptor.) The number of bytes to copy is in DD.SIZ. Use the drive number (PD.DRV) to compute the offset for the corresponding drive table as follows:

OS-9 Technical Reference

LDA PD.DRV,Y	Get the drive number
LDB #DRVMEN	Get the size of a drive table
MUL	
LEAX DRVBEG,U	Get the address of the first table
LEAX D,X	Compute the address of the table

Write Writes a 256-byte sector buffer to a disk.

Entry Conditions:

- B = *MSB of the disk LSN*
- X = *LSB of the disk LSN*
- Y = *address of the path descriptor*
- U = *address of the device memory area*

Exit Conditions:

- CC = *carry set on error*
- B = *error code*

Additional Information:

- Following is a typical routine for using Write:
 1. Get the sector buffer address from PD.BUF in the path descriptor.
 2. Get the drive number from PD.DRV in the path descriptor.
 3. Compute the physical disk address from the logical sector number.
 4. Initiate the Write operation.
 5. Copy V.BUSY to V.WAKE. The driver then goes to sleep and waits for the I/O to complete. (The IRQ service routine sends the wakeup signal.) After awakening, the driver tests V.WAKE to see if it is clear. If it is not, the driver goes back to sleep. If the disk controller cannot be interrupt-driven, it is necessary to perform a programmed I/O transfer.
 6. If PF.VFY in the path descriptor is equal to zero, read the sector back in and verify that it is written correctly. Verification usually does not involve a comparison of all of the data bytes.
- If disk writes are to be verified, the Init routine must request the buffer in which to place the sector when it is read back. Do not copy LSN 0 into the drive table when reading it back for verification.

- Use the drive number (PD.DRV) to compute the offset to the corresponding drive table as shown for the Read routine.

Getstats and Setstats

Reads or changes device's operating parameters.

Entry Conditions:

U = address of the device memory area
Y = address of the path descriptor
A = status code

Exit Conditions:

B = error code (if any)
CC = carry set on error

Additional Information:

- Get/set the device's operating parameters (status) as specified for the Get Status and Set Status system calls. Getsta and Setsta are wild card calls.
- It might be necessary to examine or change the register stack that contains the values of the 6809 register at the time of the call. The address of the register stack is in PD.RGS, which is located in the path descriptor. You can use the following offsets to access any value in the register stack:

Reg.	Relative Addr.	Size	6809 Reg.
R\$CC	\$00	1	Condition Code Reg.
R\$D	\$01	2	Register D
R\$A	\$01	1	Register A
R\$B	\$02	1	Register B
R\$DP	\$03	1	Register DP
R\$X	\$04	2	Register X
R\$Y	\$06	2	Register Y
R\$U	\$08	2	Register U
R\$PC	\$0A	2	Program Counter

- Register D overlays Registers A and B.

Term Terminate a device.

Entry Conditions:

U = *address of the device memory area*

Exit Conditions:

CC = *carry set on error*

B = *error code (if any)*

Additional Information:

- This routine is called when a device is no longer in use in the system (when the link count of its device descriptor module becomes zero).
- Following is a typical routine for using Term:
 1. Wait until any pending I/O is completed.
 2. Disable the device interrupts.
 3. Remove the device from the IRQ polling list.
 4. If the Init routine reserved a 256-byte buffer for verifying disk writes, return the memory with the Return System system call (F\$SRtMem).

IRQ Service Routine

Services device interrupts.

Additional Information:

- The IRQ Service routine sends a wakeup signal to the process indicated by the process ID in V.WAKE when the I/O is complete. It then clears V.WAKE as a flag to indicate to the main program that the IRQ has indeed occurred.
- When the IRQ service routine finishes servicing an interrupt it must clear the carry and exit with an RTS instruction.
- Although this routine is not included in the device driver module branch table and is not called directly by the RBF manager, it is a key routine in interrupt-driven drivers. Its function is to:
 1. Service the device interrupts (receive data from device or send data to it). The IRQ service routine puts its data into and get its data from buffers that are defined in the device memory area.
 2. Wake up a process that is waiting for I/O to be completed. To do this, the routine checks to see if there is a process ID in V.WAKE (if the bit is non-zero); if so, it sends a wakeup signal to that process.
 3. If the device is ready to send more data, and the output buffer is empty, disable the device's *ready to transmit* interrupts.

Boot (Bootstrap Module)

Loads the boot file into RAM.

Entry Conditions:

None

Exit Conditions:

- D = size of the boot file (in bytes)
- X = address at which the boot file was loaded into memory
- CC = carry set on error
- B = error code (if any)

Additional Information:

- The Boot module is not part of the disk driver. It is a separate module that is stored on the boot track of the system disk with OS9P1 and REL.
- The bootstrap module contains one subroutine that loads the bootstrap file and related information into memory. It uses the standard executable module format with a module type of \$C. The execution offset in the module header contains the offset to the entry point of this subroutine.
- The module gets the starting sector number and size of the OS9Boot file from LSN 0. OS-9 allocates a memory area large enough for the Boot file. Then, it loads the Boot file into this memory area.
- Following is a typical routine for using Boot:
 1. Read LSN 0 from the disk into a buffer area. The Boot module must pick its own buffer area. LSN 0 contains the values for DD.BT (the 24-bit LSN of the bootstrap file), and DD.BSZ (the size of the bootstrap file in bytes).
 2. Get the 24-bit LSN of the bootstrap file from DD.BT.
 3. Get the size of the bootstrap file from DD.BSZ. The Boot module is contained in one logically contiguous block beginning at the logical sector specified in DD.BT and extending for $DD.BSZ/256 + 1$ sectors.

4. Use the OS-9 Request Systemem system call (F\$SRqMem) to request the memory area in which the Boot file is loaded.
5. Read the Boot file into this memory area.
6. Return the size of the Boot file and its location. Boot file is loaded.

Sequential Character File Manager

The Sequential Character File Manager (SCF) supports devices that operate on a character-by-character basis. These include terminals, printers, and modems.

SCF is a re-entrant subroutine package. The I/O manager calls the SCF manager for I/O system handling of sequential, character-oriented devices. The SCF manager includes the extensive I/O editing functions typical of line-oriented operation, such as:

- backspace
- line delete
- line repeat
- auto line feed
- screen pause
- return delay padding

The SCF-type device driver modules are CC3IO, PRINTER, and RS-232. They run the video display, printer, and serial ports respectively. See the *OS-9 Commands* manual for additional Color Computer I/O devices.

SCF Line Editing Functions

The SCF manager supports two sets of read and write functions. `I$Read` and `I$Write` pass data with no modification. `I$ReadLn` and `I$WritLn` provide full line editing of device functions.

Read and Write

The Read and Write system calls to SCF-type devices correspond to the BASIC09 `GET` and `PUT` statements. While they perform little modification to the data they pass, they do filter out keyboard interrupt, keyboard terminate, and pause character. (Editing is disabled if the corresponding character in the path descriptor contains a zero.)

Carriage returns are not followed by line feeds or nulls automatically, and the high order bits are passed as sent/received.

Read Line and Write Line

The Read Line and Write Line system calls to SCF-type devices correspond to the BASIC09 INPUT, PRINT, READ, and WRITE statements. They provide full line editing of all functions enabled for a particular device.

The system initializes I\$ReadLn and I\$WritLn functions when you first use a particular device. (OS-9 copies the option table from the device descriptor table associated with the specific device.)

Later, you can alter the calls—either from assembly-language programs (using the Get Status system call), or from the keyboard (using the TMODE command). All bytes transferred by ReadLn and WritLn have the high order bit cleared.

SCF Definitions of the Path Descriptor

The PD.FST and PD.OPT sections of the path descriptor are reserved for and used by the SCF file manager.

The following table describes the SCF manager's use of PD.FST and PD.OPT. For your convenience, the table also includes the other sections of the PD.

The PD.OPT section contains the values that determine the line editing functions. It contains many device operating parameters that can be read or written by the Set Status or Get Status system call. Any values not set by this table default to zero.

Note: You can disable most of the editing functions by setting the corresponding control character in the path descriptor to zero. You can use the Set Status system call or the TMODE command to do this. Or, you can go a step further by setting the corresponding control character value in the device descriptor module to zero.

To determine the default settings for a specific device, you can inspect the device descriptor.

Name	Relative Address	Size (Bytes)	Use
Universal Section (Same for all file managers)			
PD.PD	\$00	1	Path number
PD.MOD	\$01	1	Access mode: 1 = read 2 = write 3 = update
PD.CNT	\$02	1	Number of open images (paths using this PD)
PD.DEV	\$03	2	Address of the associated device table entry
PD.CPR	\$05	1	Current process ID
PD.RGS	\$06	2	Address of the caller's 6809 register stack
PD.BUF	\$08	2	Address of the 256-byte data buffer (if used)

Name	Relative Address	Size (Bytes)	Use
SCF Path Descriptor Definitions (PD.FST Section)			
PD.DV2	\$0A	2	Device table address of the second (echo) device
PD.RAW	\$0C	1	Edit flag: 0 = raw mode 1 = edit mode
PD.MAX	\$0D	2	Read Line maximum character count
PD.MIN	\$0F	1	Devices are <i>mine</i> if cleared
PD.STS	\$10	2	Status routine module address
PD.STM	\$12	2	Reserved for status routine

Name	Relative Address	Size (Bytes)	Use
SCF Option Section Definition (PD.OPT Section)			
(Copied from the device descriptor)			
PD.DTP	\$20	1	Device class: 0 = SCF 1 = RBF 2 = PIPE 3 = SBF
PD.UPC	\$21	1	Case: 0 = upper and lower 1 = upper only
PD.BSO	\$22	1	Backspace: 0 = backspace 1 = backspace, space and backspace
PD.DLO	\$23	1	Delete: 0 = backspace over line 1 = carriage return, line feed
PD.EKO	\$24	1	Echo: 0 = no echo
PD.ALF	\$25	1	Auto line feed: 0 = no auto line feed
PD.NUL	\$26	1	End-of-line null count: n = number of nulls (\$00) sent after each carriage return or carriage return and line feed (n = \$00-\$FF)
PD.PAU	\$27	1	End of page pause: 0 = no pause
PD.PAG	\$28	1	Number of lines per page
PD.BSP	\$29	1	Backspace character
PD.DEL	\$2A	1	Delete-line character

Name	Relative Address	Size (Bytes)	Use
SCF Option Section Definition continued (PD.OPT Section)			
PD.EOR	\$2B	1	End-of-record character (End-of-line character) Read only. Normally set to \$0D: 0 = Terminate read-line only at the end of the file
PD.EOF	\$2C	1	End-of-file character (read only)
PD.RPR	\$2D	1	Reprint-line character
PD.DUP	\$2E	1	Duplicate-last-line character
PD.PSC	\$2F	1	Pause character
PD.INT	\$30	1	Keyboard-interrupt character
PD.QUT	\$31	1	Keyboard-terminate character
PD.BSE	\$32	1	Backspace-echo character
PD.OVF	\$33	1	Line-overflow character (bell CTRL G)
PD.PAR	\$34	1	Device-initialization value (parity)
PD.BAU	\$35	1	Software settable baud rate
PD.D2P	\$36	2	Offset to second device name string
PP.XON	\$38	1	ACIA XON char
PD.XOFF	\$39	1	ACIA XOFF char
PD.ERR	\$3A	1	Most recent I/O error status
PD.TBL	\$3B	2	Copy of device table address
PD.PLP	\$3D	2	Path descriptor list pointer
PD.PST	\$3F	1	Current path status

PD.EOF specifies the end-of-file character. If this is the first and only character that is input to the SCF device, SCF returns an end-of-file error on Read or Readln.

PD.PSC specifies the pause character, which suspends output to the device before the next end-of-record character. The pause character also deletes any type-ahead input for Readln.

PD.INT specifies the keyboard-interrupt character. When the character is received, the system sends a keyboard terminate signal to the last user of a path. The character also terminates the current I/O request (if any) with an error identical to the keyboard interrupt signal code.

PD.QUT specifies the keyboard-terminate character. When this character is received, the system sends a keyboard terminate signal to the last user of a path. The system also cancels the current I/O request (if any) by sending error code identical to the keyboard interrupt signal code.

PD.PAR specifies the parity information for external serial devices.

PD.BAU specifies baud rate, word length and stop bit information for serial devices.

PD.XON contains either the character used to enable transmission of characters or a null character that disables the use of XON.

PD.XOFF contains either the character used to disable transmission of characters or a null character that disables the use of XOFF.

SCF-Type Device Descriptor Modules

The following chart shows how the initialization table in the device descriptors is used for SCF-type devices. The values are those the I/O manager copies from the device descriptor to the path descriptor.

An SCF editing function is turned off if its corresponding value is set to zero. For example, if IT.EOF is set to zero, there is no end-of-file character.

Name	Relative Address	Size (Bytes)	Use
(header)	\$00-11		Standard device descriptor module header
IT.DVC	\$12	1	Device class: 0 = SCF 1 = RBF 2 = PIPE 3 = SBF
IT.UPC	\$13	1	Case: 0 = upper- and lowercase 1 = uppercase only
IT.BSO	\$14	1	Backspace: 0 = backspace 1 = backspace, then space and backspace
IT.DLO	\$15	1	Delete: 0 = backspace over line 1 = carriage return
IT.EKO	\$16	1	Echo: 0 = echo off
IT.ALF	\$17	1	Auto line feed: 0 = auto line feed disabled
IT.NUL	\$18	1	End-of-line null count
IT.PAU	\$19	1	Pause: 0 = end-of-page pause disabled
IT.PAG	\$1A	1	Number of lines per page
IT.BSP	\$1B	1	Backspace character
IT.DEL	\$1C	1	Delete-line character
IT.EOR	\$1D	1	End-of-record character
IT.EOF	\$1E	1	End-of-file character
IT.RPR	\$1F	1	Reprint-line character

Name	Relative Address	Size (Bytes)	Use
IT.DUP	\$20	1	Duplicate-last-line character
IT.PSC	\$21	1	Pause character
IT.INT	\$22	1	Interrupt character
IT.QUT	\$23	1	Quit character
IT.BSE	\$24	1	Backspace echo character
IT.OVF	\$25	1	Line-overflow character (bell)
IT.PAR	\$26	1	Initialization value—used to initialize a device control register when a path is opened to it (parity)
IT.BAU	\$27	1	Baud rate
IT.D2P	\$28	2	Attached device name string offset
IT.XON	\$2A	1	X-ON character
IT.XOFF	\$2B	1	X-OFF character
IT.COL	\$2C	1	Number of columns for display
IT.ROW	\$2D	1	Number of rows for display
IT.WND	\$2E	1	Window number
IT.VAL	\$2F	1	Data in rest of descriptor is valid
IT.STY	\$30	1	Window type
IT.CPX	\$31	1	X cursor position
IT.CPY	\$32	1	Y cursor position
IT.FGC	\$33	1	Foreground color
IT.BGC	\$34	1	Background color
IT.BDC	\$35	1	Border color

SCF-Type Device Driver Modules

An SCF-type device driver module contains a package of subroutines that perform raw (unformatted) data I/O transfers to or from a specific hardware controller. Such a module is usually re-entrant so that one copy of the module can simultaneously run several devices that use identical I/O controllers. The I/O manager allocates a permanent memory area for each controller sharing the driver.

The size of the memory area is defined in the device driver module header. The I/O manager and SCF use some of the memory area. The device driver can use the rest in any way (typically as variables and buffers). Typically, the driver uses the area as follows:

Name	Relative Address	Size (Bytes)	Use
V.PAGE	\$00	1	Port extended 24 bit address
V.PORT	\$01	2	Device base address (defined by the I/O manager)
V.LPRC	\$03	1	ID of the last active process
V.BUSY	\$04	1	ID of the active process (defined by RBF): 0 = no active process
V.WAKE	\$05	1	ID of the process to reawaken after the device completes I/O (defined by the device driver): 0 = no waiting process
V.USER	\$06	0	Beginning of file manager specific storage
V.TYPE	\$06	1	Device type or parity
V.LINE	\$07	1	Lines left until the end of the page
V.PAUS	\$08	1	Pause request: 0 = no pause requested
V.DEV2	\$09	2	Attached device memory area
V.INTR	\$0B	1	Interrupt character

Name	Relative Address	Size (Bytes)	Use
V.QUIT	\$0C	1	Quit character
V.PCHR	\$0D	1	Pause character
V.ERR	\$0E	1	Error accumulator
V.XON	\$0F	1	XON character
V.XOFF	\$10	1	XOFF character
V.KANJI	\$11	1	Reserved
V.KBUF	\$12	2	Reserved
V.MODADR	\$14	2	Reserved
V.PDLHD	\$16	2	Path descriptor list header
V.RSV	\$18	5	Reserved
V.SCF	\$1D	0	End of SCF memory requirements
FREE	\$1D	0	Free for the device driver to use

V.LPRC contains the process ID of the last process to use the device. The IRQ service routine sends this process the proper signal if it receives a quit character or an interrupt character. V.LPRC is defined by SCF.

V.BUSY contains the process ID of the process that is using the device. (If the device is not being used, V.BUSY contains a zero.) The process ID is used by SCF to prevent more than one process from using the device at the same time. V.BUSY is defined by SCF.

SCF Device Driver Subroutines

Like all device drivers, SCF device drivers use a standard executable memory module format.

The execution offset address in the module header points to a branch table that has six 3-byte entries. Each entry is typically an LBRA to the corresponding subroutine. The branch table is defined as follows:

ENTRY	LBRA	INIT	Initialize driver
	LBRA	READ	Read character
	LBRA	WRITE	Write character
	LBRA	GETSTA	Get status
	LBRA	SETSTA	Set status
	LBRA	TERM	Terminate device

If no error occurs, each subroutine exits with the C bit in the Condition Code Register cleared. If an error occurred, each subroutine sets the C bit and returns an appropriate error code in Register B.

The rest of this chapter describes these subroutines and their entry and exit conditions.

Init Initializes device control registers, and enables interrupts if necessary.

Entry Conditions:

- Y = address of the device descriptor
- U = address of the device memory area

Exit Conditions:

- CC = carry set on error
- B = error code (if any)

Additional Information:

- Prior to being called, the device memory area is cleared (set to zero), except for V.PAGE and V.PORT. (V.PAGE and V.PORT contain the device address.) There is no need to initialize the part of the memory area used by the I/O manager and SCF.
- Follow these steps to use Init:
 1. Initialize the device memory area.
 2. Place the IRQ service routine on the IRQ polling list, using the Set IRQ system call (F\$IRQ).
 3. Initialize the device control registers.

Read Reads the next character from the input buffer.

Entry Conditions:

Y = *address of the path descriptor*
U = *address of the device memory area*

Exit Conditions:

A = *character read*
CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- This is a step by step description of a Read operation:
 1. Read gets the next character from the input buffer.
 2. If no data is ready, Read copies its process ID from V.BUSY into V.WAKE. It then uses the Sleep system call to put itself to sleep.
 3. Later, when Read receives data, the IRQ service routine leaves the data in a buffer. Then, the routine checks V.WAKE to see if any process is waiting for the device to complete I/O. If so, the IRQ service routine sends a wakeup signal to the waiting process.
- Data buffers are not automatically allocated. If a buffer is used, it defines it in the device memory area.

Write Sends a character (places a data byte in an output buffer) and enables the device output interrupts.

Entry Conditions:

- A = *character to write*
- Y = *address of the path descriptor*
- U = *address of the device memory area*

Exit Conditions:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- If the data buffer is full, Write copies its process ID from V.BUSY into V.WAKE. Write then puts itself to sleep.

Later, when the IRQ service routine transmits a character and makes room for more data, it checks V.WAKE to see if there is a process waiting for the device to complete I/O. If there is, the routine sends a wakeup signal to that process.

- Write must ensure that the IRQ service routine that starts it begins to place data in the buffer. After an interrupt is generated, the IRQ service routine continues to transmit data until the data buffer is empty. Then, it disables the device's ready-to-transmit interrupts.
- Data buffers are not allocated automatically. If a buffer is used, define it in the device memory area.

Getsta and Setsta

Gets/sets device operating parameters (status) as specified for the Get Status and Set Status system calls. Getsta and Setsta are wildcard calls.

Entry Conditions:

- A = depends on the function code
- Y = address of the path descriptor
- U = address of the device memory area
- Other registers depend on the function code.

Exit Conditions:

- B = error code (if any)
- CC = carry set on error
- Other registers depend on the function code

Additional Information:

- Any codes not defined by the I/O manager or SCF are passed to the device driver.
- You might need to examine or change the register stack that contains the values of the 6809 registers at the time of the call. The address of the register stack can be found in PD.RGS, which is located in the path descriptor.
- You can use the following offsets to access any value in the register packet:

Name	Relative Address	Size (Bytes)	6809 Register
R\$CC	\$00	1	Condition Codes Register
R\$D	\$01	0	Register D
R\$A	\$01	1	Register A
R\$B	\$02	1	Register B
R\$DP	\$03	1	Register DP
R\$X	\$04	2	Register X
R\$Y	\$06	2	Register Y
R\$U	\$08	2	Register U
R\$PC	\$0A	2	Program Counter

The function code is retrieved from the R\$B on the user stack.

Term Terminates a device. Term is called when a device is no longer in use (when the link count of the device descriptor module becomes zero).

Entry Conditions:

U = pointer to the device memory area

Exit Conditions:

CC = carry set on error

B = error code (if any)

Additional Information:

- To use Term:
 1. Wait until the IRQ service routine empties the output buffer.
 2. Disable the device interrupts.
 3. Remove the device from the IRQ polling list.
- When Term closes the last path to a device, OS-9 returns to the memory pool the memory that the device used. If the device has been attached to the system using the I\$Attach system call, OS-9 does not return the static storage for the driver until an I\$Detach call is made to the device. Modules contained in the Boot file are never terminated, even if their link counts reach 0.

IRQ Service Routine

Receives device interrupts. When I/O is complete, the routine sends a wakeup signal to the process identified by the process ID in V.WAKE. The routine also clears V.WAKE as a flag to indicate to the main program that the IRQ has occurred.

Additional Information:

- The IRQ Service Routine is not included in device driver branch tables, and is not called directly by SCF. However, it is a key routine in device drivers.
- When the IRQ Service routine finishes servicing an interrupt, the routine must clear the carry and exit with an RTS instruction.
- Here is a typical sequence of events that the IRQ Service Routine performs:
 1. Service the device interrupts (receive data from the device or send data to it). Ensure this routine puts its data into and get its data from buffers that are defined in the device memory area.
 2. Wake up any process that is waiting for I/O to complete. To do this, the routine checks to see if there is a process ID in V.WAKE (a value other than zero); if so, it sends a wakeup signal to that process.
 3. If the device is ready to send more data, and the output buffer is empty, disable the device's ready-to-transmit interrupts.
 4. If a pause character is received, set V.PAUS in the attached device storage area to a value other than zero. The address of the attached device memory area is in V.DEV2.

V.PAUS in the attached device storage area to non-zero value. The address of the attached device memory area is in V.DEV2.
 5. If a keyboard terminate or interrupt character is received, signal the process in V.LPRC (last known process) if any.

The Pipe File Manager (PIPEMAN)

The Pipe file manager handles control of processes that use paths to pipes. Pipes allow concurrently executing processes to send each other data by using the output of one process (the writer) as input to a second process (the reader). The reader gets input from the standard input. The exclamation point (!) operator specifies that the input or output is from or to a pipe. The Pipe file manager allocates a 256-byte block and a path descriptor for data transfer. The Pipe file manager also determines which process has control of the pipe. The Pipe file manager has the standard file manager branch table at its entry point:

```
PipeEnt  lbra Create
         lbra Open
         lbra MakDir
         lbra ChgDir
         lbra Delete
         lbra Seek
         lbra PRead
         lbra PWrite
         lbra PRdLn
         lbra PWrLn
         lbra Getstat
         lbra Putstat
         lbra Close
```

You cannot use MakDir, ChgDir, Delete, and Seek with pipes. If you try to do so, the system returns E\$UNKSVC (unknown service request). Getstat and Putstat are also no-action service routines. They return without error.

Create and Open perform the same functions. They set up the 256-byte data exchange buffer, and save several addresses in the path descriptor.

The Close request checks to see if any process is reading or writing through the pipe. If not, OS-9 returns the buffer.

PRead, PWrite, PRdLn, and PWrLn read data from the buffer and write data to it.

The ! operator tells the Shell that processes wish to communicate through a pipe. For example:

```
proc1 ! proc2 
```

In this example, shell forks Proc1 with the standard output path to a pipe, and forks Proc2 with the Standard input path from a pipe.

Shell can also handle a series of processes using pipes. Example:

```
proc1 ! proc2 ! proc3 ! proc4 
```

The following outline shows how to set up pipes between processes:

Open /pipe	save path in variable x
Dup path #1	save stdout in variable y
Dup x	make path available
Fork proc1	put pipe in stdout (Dup uses lowest available)
Close #1	make path available
Dup y	restore stdout
Close y	make path available
Dup path #0	save stdin in Y
Close #0	make path available
Dup x	put pipe in stdin
Fork 2	fork process 2
Close #0	make path available
Dup y	restore stdin
Close x	no longer needed
Close y	no longer needed

Example: The following example shows how an application can initiate another process with the stdin and stdout routed through a pipe.

```
Open /pipe1      save path in variable a
Open /pipe2      save path in variable b
Dup 0            save stdin in variable x
Dup 1            save stdout in variable y
Close 0          make path available
Close 1          make path available
Dup a            make pipe1 stdin
Dup b            make pipe2 stdout
Fork new process
Close 0          make path available
Close 1          make path available
Dup x            restore stdin
Dup y            restore stdout
Return a&b       return pipe path numbers to caller
```


System Calls

System calls are used to communicate between the OS-9 operating system and assembly-language programs. There are two major types of calls—I/O calls and function calls.

Function calls include user mode calls and system mode calls.

Each system call has a mnemonic name. Names of I/O calls start with I\$. For example, the Change Directory call is I\$ChgDir. Names of function calls start with F\$. For example, the Allocate Bits call is F\$AllBit. The names are defined in the assembler-input conditions equate file called OS9Defs.

System mode calls are privileged. You can execute them only while OS-9 is in the system state (when it is processing another system call, executing a file manager or device driver, and so on).

System mode calls are included in this manual primarily for programmers writing device drivers and other system-level applications.

Calling Procedure

To execute any system calls, you need to use an SWI2 instruction:

1. Load the 6809 registers with any appropriate parameters.
2. Execute an SWI2 instruction, followed immediately by a constant byte, which is the request code. In the references in this chapter, the first line is the system call name (for example Close Path) and the second line contains the call's mnemonic name (for example I\$Close), the software interrupt Code 2 (103F), and the call's request code (for example, 8F) in hexadecimal.
3. After OS-9 processes the call, it returns any parameters in the 6809 registers. If an error occurs, the C bit of the condition code register is set, and Register B contains the appropriate error code. This feature permits a BCS or BCC instruction immediately following the system call to branch either if there is an error or if no error occurs.

As an example, here is the Close system call:

```
LDA PATHNUM
SWI2
FCB $8F
BCS ERROR
```

You can use the assembler's *OS9 directive* to simplify the call, as follows.

```
LDA PATHNUM
OS9 I$Close
BCS ERROR
```

The ASM assembler allows any combination of upper- or lower-case letters. The RMA assembler, included in the OS-9 Level Two Development Pak, is case sensitive. The names in this manual have been spelled with upper and lower case letters, matching the defs for RMA.

I/O System Calls

OS-9's I/O calls are easier to use than many other systems' I/O calls. This is because the calling program does not have to allocate and set up *file control blocks*, *sector buffers*, and so on.

Instead, OS-9 returns a 1-byte path number whenever a process opens a path to a file or device. Until the path is closed, you can use this path number in later I/O requests to identify the file or device.

In addition, OS-9 allocates and maintains its own data structures; so, you need not deal with them.

System Call Descriptions

The rest of this chapter consists of the system call descriptions. At the top of each description is the system call name, followed by its mnemonic name, the SWI2 code and the request code. Next are the call's entry and exit conditions, followed by additional information about the code where appropriate.

In the system call descriptions, registers not specified as entry or exit conditions are not altered. Strings passed as parameters are normally terminated with a space character and end-of-line character, or with Bit 7 of the last character set.

If an error occurs on a system call, the C bit of Register CC is set, and Register B contains the *error code*. If no error occurs, the C bit is clear, and Register B contains a value of zero.

User Mode System Calls Quick Reference

Following is a summary of the User Mode System Calls referenced in this chapter:

F\$AllBit	Sets bits in an allocation bit map
F\$Chain	Chains a process to a new module
F\$CmpNam	Compares two names
F\$CpyMem	Copies external memory
F\$CRC	Generates a cyclic redundancy check
F\$DelBit	Deallocates bits in an allocation bit map
F\$Exit	Terminates a process
F\$Fork	Starts a new process
F\$GBlkMp	Gets a copy of a system block map
F\$GModDr	Gets a copy of a module directory
F\$GPrDsc	Gets a copy of a process descriptor
F\$Icpt	Sets a signal intercept trap
F\$ID	Returns a process ID
F\$Link	Links to a memory module
F\$Load	Loads a module from mass storage
F\$Mem	Changes a process's data area size
F\$NMLink	Links to a module; does not map the module into the user's address space
F\$NMLoad	Loads a module but does not map it into the user's address space
F\$Perr	Prints an error message
F\$PrsNam	Parses a pathlist name
F\$SchBit	Searches a bit map

F\$Send	Sends a signal to a process
F\$Sleep	Suspends a process
F\$SPrior	Sets a process's priority
F\$SSWI	Sets a software interrupt vector
F\$STime	Sets the system time
F\$SUser	Sets the user ID number
F\$Time	Returns the current time
F\$UnLink	Unlinks a module
F\$UnLoad	Unlinks a module by name
F\$Wait	Waits for a signal
I\$Attach	Attaches an I/O device
I\$Chgdir	Changes a working directory
I\$Close	Closes a path
I\$Create	Creates a new file
I\$Delete	Deletes a file
I\$DeletX	Deletes a file from the execution directory
I\$Detach	Detaches an I/O device
I\$Dup	Duplicates a path
I\$GetStt	Gets a device's status
I\$MakDir	Creates a directory file
I\$Open	Opens a path to an existing file
I\$Read	Reads data from a device
I\$ReadLn	Reads a line of data from a device
I\$Seek	Positions a file pointer
I\$SetStt	Sets a device's status
I\$Write	Writes data to a device
I\$WritLn	Writes a data line to a device

System Mode Calls Quick Reference

Following is a summary of the System Mode Calls referenced in this chapter:

F\$Alarm	Sets up an alarm
F\$All64	Allocates a 64-byte memory block
F\$AllHRAM	Allocates high RAM
F\$AllImg	Allocates image RAM blocks
F\$AllPrc	Allocates a process descriptor
F\$AllRAM	Allocates RAM blocks
F\$AllTsk	Allocates a process task number
F\$AProc	Enters active process queue
F\$Boot	Performs a system bootstrap
F\$BtMem	Performs a memory request bootstrap
F\$ClrBlk	Clears the specified block of memory
F\$DATLog	Converts a DAT block offset to a logical address
F\$DelImg	Deallocates image RAM blocks
F\$DelPrc	Deallocates a process descriptor
F\$DelRAM	Deallocates RAM blocks
F\$DelTsk	Deallocates a process task number
F\$ELink	Links modules using a module directory entry
F\$FModul	Finds a module directory entry
F\$Find64	Finds a 64-byte memory block
F\$FreeHB	Gets a free high block
F\$FreeLB	Gets a free low block
F\$GCMDir	Compacts module directory entries
F\$GProcP	Gets a process's pointer

F\$IODel	Deletes an I/O module
F\$IOQu	Puts an entry into an I/O queue
F\$IRQ	Makes an entry into IRQ polling table
F\$LDABX	Loads Register A from 0,X in Task B
F\$LDAXY	Loads A[X,[Y]]
F\$LDDDXY	Loads D[D + X,[Y]]
F\$MapBlk	Maps the specified block
F\$Move	Moves data to a different address space
F\$NProc	Starts the next process
F\$RelTsk	Releases a task number
F\$ResTsk	Reserves a task number
F\$Ret64	Returns a 64-byte memory block
F\$SetImg	Sets a process DAT image
F\$SetTsk	Sets a process's task DAT registers
F\$SLink	Performs a system link
F\$SRqMem	Performs a system memory request
F\$SRtMem	Performs a system memory return
F\$SSvc	Installs a function request
F\$STABX	Stores Register A at 0,x in Task B
F\$VIRQ	Makes an entry in a virtual IRQ polling table
F\$VModul	Validates a module

User System Calls

Allocate Bits

OS9 F\$AllBit 103F 13

Sets bits in an
allocation bit map

Entry Conditions:

- D = *number of the first bit to set*
- X = *starting address of the allocation bit map*
- Y = *number of bits to set*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- Bit numbers range from 0 to $n-1$, where n is the number of bits in the allocation bit map.
- **Warning:** Do not issue the Allocate Bits call with Register Y set to 0 (a bit count of 0).

Chain

OS9 F\$Chain 103F 05

Loads and executes a new primary module without creating a new process

Entry Conditions:

- A = *language/type code*
- B = *size of the data area* (in pages); must be at least one page
- X = *address of the module name or filename*
- Y = *parameter area size* (in bytes); defaults to zero if not specified
- U = *starting address of the parameter area*

Error Output:

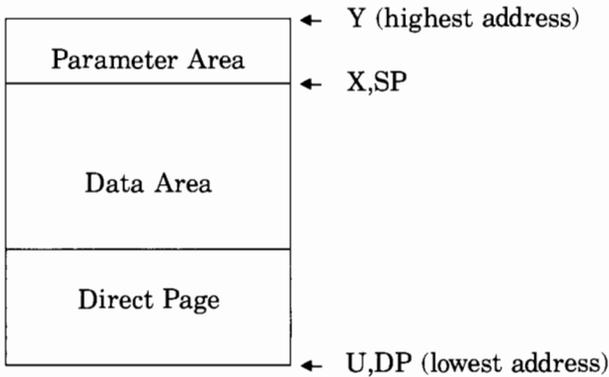
- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- Chain loads and executes a new primary module, but does not create a new process. A Chain system call is similar to a Fork followed by an Exit, but it has less processing overhead. Chain resets the calling process program and data memory areas and begins executing a new primary module. It does not affect open paths. This is a user mode system call.
- **Warning:** Make sure that the hardware stack pointer (Register SP) is located in the direct page before Chain executes. Otherwise the system might crash or return a suicide attempt error. This precaution also prevents a suicide in the event that the new module requires a smaller data area than that in use. Allow approximately 200 bytes of stack space for execution of the Chain system call.
- Chain performs the following steps:
 1. It causes OS-9 to unlink the process's old primary module.

2. OS-9 parses the name string of the new process's primary module (the program that is to be executed first). Then, it causes OS-9 to search the system module directory to see if a module with the same name, type, and language is already in memory.
3. If the module is in memory, it links to it. If the module is not in memory, it uses the name string as the path-list of a file to load into memory. Then, it links to the first module in this file. (Several modules can be loaded from a single file.)
4. It reconfigures the data memory area to the size specified in the new primary module's header.
5. It intercepts and erases any pending signals.

The following diagram shows how Chain sets up the data memory area and registers for the new module.



- D = *parameter area size*
- PC = *module entry point absolute address*
- CC = F=0, I=0; others are undefined

Registers Y and U (the top-of-memory and bottom-of-memory pointers, respectively) always have values at page boundaries. If the parent process does not specify a size for the parameter area, the size (Register D) defaults to zero. The data area must be at least one page long.

(For more information, see the Fork system call.)

Compare Names

Compares two strings
for a match

OS9 F\$CmpNam 103F 11

Entry Conditions:

B = length of string1
X = address of string1
Y = address of string2

Exit Conditions:

CC = carry clear if the strings match

Additional Information:

- The Compare Names call compares two strings and indicates whether they match. Use this call with the Parse Name system call. The second string must have the most significant bit (Bit 7) of the last character set.

Copy External Memory

Reads external memory into the user's buffer for inspection

OS9 F\$CpyMem
103F 1B

Entry Conditions:

- D = *DAT image pointer*
- X = *offset in block to begin copy*
- Y = *byte count*
- U = *caller's destination buffer*

Error Output:

- CC = *C bit set on error*
- B = *appropriate error code*

Additional Information:

- You can view any system memory through the use of the Copy External Memory call. The call assumes X is the address of the 64K space described by the DAT image given.
- If you pass the entire DAT image of a process, place a value in the X Register that equals the address in the process space. If you pass a partial DAT image (the upper half), then place a value in Register X that equals the offset from the beginning of the DAT image (\$8000).
- The support module for this call is OS9p2.

CRC

Calculates the CRC of
a module

OS9 F\$CRC 103F 17

Entry Conditions:

- X = *starting byte address*
- Y = *number of bytes*
- U = *address of the 3-byte CRC accumulator*

Exit Conditions:

Updates the CRC accumulator.

Additional Information:

- The CRC call calculates the CRC (cyclic redundancy count) for use by compilers, assemblers, or other module generators.
- The calculation begins at the *starting byte address* and continues over the specified *number of bytes*.
- You need not cover an entire module in one call, since the CRC can be accumulated over several calls. The CRC accumulator can be any 3-byte memory area. You must initialize it to \$FFFFFF before the first CRC call.
- When checking an existing module CRC, the calculation should be performed on the entire module (including the module CRC). The CRC accumulator will contain the CRC constant bytes if the module CRC is correct.
- If the CRC of a new module is to be generated, the CRC is accumulated over the module (excluding CRC). The accumulated CRC is complemented then stored in the correct position in the module.
- Be sure to initialize the CRC accumulator only once for each module checked by CRC.

Deallocate Bits

Clears allocation map bits

OS9 F\$DelBit 103F 14

Entry Conditions:

- D = *number of the first bit to set*
- X = *starting address of the allocation bit map*
- Y = *number of bits to set*

Exit Conditions: None

Additional Information:

- The Deallocate Bits call clears bits in the allocation bit map pointed to by Register X. Bit numbers are in the range 0 to $n-1$, where n is the number of bits in the allocation bit map.
- **Warning:** Do not call Deallocate Bits with Register Y set to 0 (a bit count of 0).

Exit

Terminates the calling process

OS9 F\$Exit 103F 06

Entry Conditions:

B = *status code to return to the parent*

Exit Conditions:

The process is terminated.

Additional Information:

- The Exit system call is the only way a process can terminate itself. Exit deallocates the process's data memory area, and unlinks the process's primary module. It also closes all open paths automatically.
- The Wait system call always returns to the parent the status code passed by the child in its Exit call. Therefore, if the parent executes a Wait and receives the status code, it knows the child has died. This is a user mode system call.
- Exit unlinks only the primary module. Unlink any module that is loaded or linked to by the process before calling Exit.

Fork

Creates a child process

OS9 F\$Fork 103F 03

Entry Conditions:

- A = *language/type code*
- B = *size of the optional data area (in pages)*
- X = *address of the module name or filename (See the following example.)*
- Y = *size of the parameter area (in pages); defaults to zero if not specified*
- U = *starting address of the parameter area; must be at least one page*

Exit Conditions:

- X = *address of the last byte of the name + 1 (See the following example.)*
- A = *new process IO number*

Error Output:

- B = *error code (if any)*
- CC = *carry set on error*

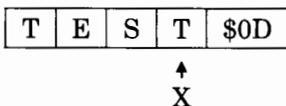
Additional Information:

- Fork creates a new process, a child of the calling process. Fork also sets up the child process's memory and 6809 registers and standard I/O paths.
- Before the Fork call:

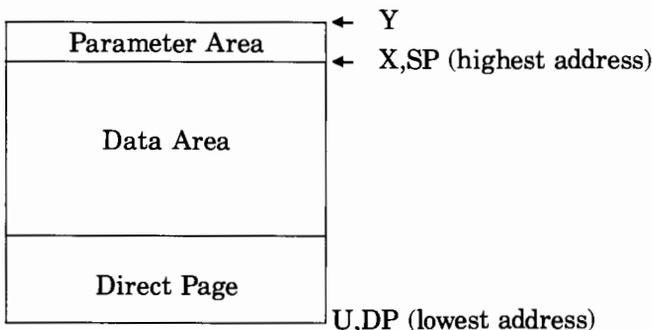
T	E	S	T	\$0D
---	---	---	---	------

↑
X

- After the Fork call:



- This is the sequence of Fork's operations:
 1. OS-9 parses the name string of the new process's primary module (the program that OS-9 executes first). Then, it searches the system module directory to see if the program already is in memory.
 - 2a. The next step depends on whether or not the program is already in memory. If the program is in memory, OS-9 links the module to the process and executes it.
 - b. If the program is not in memory, OS-9 uses the name as the pathlist of the file that is to be loaded into memory. Then, the first module in this file is linked to and executed. (Several modules can be loaded from one file.)
 3. OS-9 uses the primary module's header to determine the initial size of the process's data area. It then tries to allocate a contiguous RAM area of that size. (This area includes the parameter passing area, which is copied from the parent process's data area.)
 4. The new process's data memory area and registers are set up as shown in the following diagram. OS-9 uses the execution offset given in the module header to set the program counter to the module's entry point.



D = *size of the parameter area*
PC = *module entry point absolute address*
CC = F=0, I=0, other condition code flags are undefined

Registers Y and U (the top-of-memory pointer and bottom-of-memory pointer, respectively) always have values at page boundaries.

As stated earlier, if the parent does not specify the size of the parameter area, the size defaults to zero. The minimum overall data area size is one page.

When the shell processes a command line, it passes a string in the parameter area. This string is a copy of the parameter part of the command line. To simplify string-oriented processing, the shell also inserts an end-of-line character at the end of the parameter string.

Register X points to the start byte of the parameter string. If the command line includes the optional memory size specification (*#n* or *#nK*), the shell passes that size as the requested memory size when executing the Fork.

- If any of the preceding operations is unsuccessful, the Fork is terminated and OS-9 returns an error to the caller.
- The child and parent processes execute at the same time unless the parent executes a Wait system call immediately after the Fork. In this case, the parent waits until the child dies before it resumes execution.
- Be careful when recursively calling a program that uses the Fork system call. Another child can be created with each new execution. This continues until the process table becomes full.
- Do not fork a process with a memory size of 0.

Get System Block Map

Gets a copy of the system block map

OS9 F\$GBlkMp 103F 19

Entry Conditions:

X = *pointer to the 1024-byte buffer*

Exit Conditions:

D = *number of bytes per block (\$2000) (MMU block size dependent)*

Y = *system memory block map size*

Error Output:

CC = *carry set on error*

B = *error code (if any)*

Additional Information:

- The Get System Block Map call copies the system's memory block map into the user's buffer for inspection. The OS-9 MFREE command uses this call to find out how much free memory exists.
- The support module for this call is OS9p2.

Get Module Directory

Gets a copy of the system module directory

F\$GModDr 103F 1A

Entry Conditions:

- X = *pointer to the 2048-byte buffer*
- Y = *end of copied module directory*
- U = *start address of system module directory*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- The Get Module Directory call copies the system's module directory into the user's buffer for inspection. The OS-9 MDIR command uses this call to read the module directory.
- The support module for this call is OS9p2.

Get Process Descriptor

F\$GPrDsc 103F 18

Gets a copy of the process's process descriptor

Entry Conditions:

- A = *requested process ID*
- X = *pointer to a 512-byte buffer*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- The Get Process Descriptor call copies a process descriptor into the calling process's buffer for inspection. The data in the process descriptor cannot be changed. The OS-9 PROCS command uses this call to get information about each existing process.
- The support module for this call is OS9p2.

Intercept

Sets a signal intercept trap

OS9 F\$Icpt 103F 09

Entry Conditions:

- X = *address of the intercept routine*
- U = *starting address of the routine's memory area*

Exit Conditions:

Signals sent to the process cause the intercept routine to be called instead of the process being killed.

Additional Information:

- Intercept tells OS-9 to set a signal intercept trap. Then, whenever the process receives a signal, OS-9 executes the process's intercept routine.
- Store the address of the signal handler routine in Register X and the base address of the routine's storage area in Register U.
- Once the signal trap is set, OS-9 can execute the intercept routine at any time because a signal can occur at any time.
- Terminate the intercept routine with an RTI instruction.
- If a process has not used the Intercept system call to set a signal trap, the process terminates if it receives a signal.
- This is the order in which F\$Icpt operates:
 1. When the process receives a signal, OS-9 sets Registers U and B as follows:
 - U = *starting address of the intercept routine's memory area*
 - B = *signal code* (process's termination status)

Note: The value of Register DP cannot be the same as it was when the Intercept call was made.
 2. After setting the registers, OS-9 transfers execution to the intercept routine.

Get ID

OS9 F\$ID 103F 0C

Return's a caller's
process ID and user ID

Entry Conditions:

None

Exit Conditions:

A = *process ID*
Y = *user ID*

Additional Information:

- The *process ID* is a byte value in the range 1 to 255. OS-9 assigns each process a unique process ID.
- The *user ID* is an integer from 0 to 65535. It is defined in the system password file, and is used by the file security system and a few other functions. Several processes can have the same user ID.
- On a single user system (such as the Color Computer 3), the user ID is inherited from CC3go, which forks the initial shell.

Link

OS9 F\$Link 103F 00

Links to a memory module that has the specified name, language, and type

Entry Conditions:

- A = *type/language byte*
- X = *address of the module name* (See the following example.)

Exit Conditions:

- A = *type/language code*
- B = *attributes / revision level* (if no error)
- X = *address of the last byte of the module name + 1* (See the following example.)
- Y = *module entry point absolute address*
- U = *module header absolute address*

Error Output:

- CC = C bit set if error encountered

Additional Information:

- The module's link count increases by one whenever Link references its name. Incrementing in this manner keeps track of how many processes are using the module.
- If the module requested is not shareable (not re-entrant), only one process can link to it at a time.
- Before the Link call:

T	E	S	T	\$0D
---	---	---	---	------

↑
X

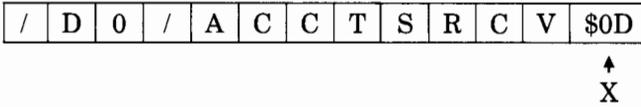
- After the Link call:

T	E	S	T	\$0D
---	---	---	---	------

↑
X

- This is the order in which the Link call operates:
 1. OS-9 searches the module directory for a module that has the specified name, language, and type.
 2. If OS-9 finds the module, the address of the module's header is returned in Register U, and the absolute address of the module's execution entry point is returned in Register Y. (This, and other information is contained in the module header.)
- If OS-9 doesn't find the module—or if the type/language codes in the entry and exit conditions don't match—OS-9 returns one of the following errors:
 - Module not found
 - Module busy (not shareable and in use)
 - Incorrect or defective module header

After the Load call:



- Possible errors:
 - Module directory full
 - Memory full
 - Errors that occur on the Open, Read, Close, and Link system calls

Memory

Changes process's data
area size

OS9 F\$Mem
103F 07

Entry Conditions:

D = *size of the new memory area* (in bytes);
0 = return current size and upper bound

Exit Conditions:

Y = *address of the new memory area upper bound*
D = *actual size of the new memory* (in bytes)

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- The memory call expands or contracts the process's data memory area to the specified size. Or, if you specify zero as the new size, the call returns the current size and upper boundaries of data memory.
- OS-9 rounds off the size to the next page boundary. In allocating additional memory, OS-9 continues upward from the previous highest address. In deallocating unneeded memory, it continues downward from that address.

Link to a module Links to a module;
OS9 F\$NMLink does not map the
103F 21 module into the user's
 address space

Entry Conditions:

- A = *type/language byte*
- X = *address of the module name*

Exit Conditions:

- A = *type/language code*
- B = *module revision*
- X = *address of the last byte of the module name + 1; any
 trailing blanks are skipped*
- Y = *storage requirement for the module*

Error Output:

- CC = *carry set on error*
- B = *error code if any*

Additional Information:

- Although this call is similar to F\$Link, it does not map the specified module into the user's address space but does return the memory requirement for the module. A calling process can use this memory requirement information to fork a program with a maximum amount of space. F\$NMLink can therefore fork larger programs than can be forked by F\$Link.

Load a module

OS9 F\$NMLoad
103F 22

Loads one or more modules from a file but does not map the module into the user's address space

Entry Conditions:

A = *type/language byte*
X = *address of the pathlist*

Exit Conditions:

A = *type/language code*
B = *module revision*
X = *address of the last byte of the pathlist + 1*
Y = *storage requirement for the module*

Error Output:

CC = *carry set on error*
B = *error code if any*

Additional Information:

- If you do not provide a full pathlist for this call, it attempts to load from a file in the current execution directory.
- Although this call is similar to F\$Load, it does not map the specified module into the user's address space but does return the memory requirement for the module. A calling process can use this memory requirement information to fork a program with a maximum amount of space. F\$NMLoad can therefore fork larger programs than can be forked by F\$Load.

Print Error

OS9 F\$Perr 103F 0F

Writes an error message to a specified path

Entry Conditions:

B = *error code*

Error Output:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- Print Error writes an error message to the standard error path for the specified process. By default, OS-9 shows:

ERROR #decimal number

- The error reporting routine is vectored. Using the Set SVC system call, you can replace it with a more elaborate reporting module.

Parse Name

Scans an input string
for a valid OS-9 name

OS9 F\$PrsNam 103F 10

Entry Conditions:

X = *address of the pathlist* (See the following example.)

Exit Conditions:

X = *address of the optional slash + 1*
Y = *address of the last character of the name + 1*
A = *trailing byte* (delimiter character)
B = *length of the name*

Error Output:

CC = *carry set*
B = *error code*
Y = *address of the first non-delimiter character in the string*

Additional Information:

- Parses, or scans, the input text string for a legal OS-9 name. It terminates the name with any character that is not a legal name character.
- Parse Name is useful for processing pathlist arguments passed to new processes.
- Because Parse Name processes only one name, you might need several calls to process a pathlist that has more than one name. As you can see from the following example, Parse Name finishes with Register Y in position for the next parse.
- If Register Y was at the end of a pathlist, Parse Name returns a bad name error. It then moves the pointer in Register Y past any space characters so that it can parse the next pathlist in a command line.

Search Bits

OS9 F\$SchBit 103F 12

Searches a specified memory allocation bit map for a free memory block of a specified size

Entry Conditions:

D = *starting bit number*
X = *starting address of the map*
Y = *bit count (free bit block size)*
U = *ending address of the map*

Error Output:

CC = C bit set

Exit Conditions:

D = *starting bit number*
Y = *bit count*

Additional Information:

- The Search Bit call searches the specified allocation bit map for a free block (cleared bits) of the required length. The search starts at the *starting bit number*. If no block of the specified size exists, the call returns with the carry set, starting bit number, and size of the largest block.

Send

Sends a signal to a specified process

OS9 F\$Send 103F 08

Entry Conditions:

- A = destination's process ID
- B = signal code

Error Output:

- CC = carry set on error
- B = error code (if any)

Additional Information:

- The *signal code* is a single byte value in the range 0 through 255.
- If the destination process is sleeping or waiting, OS-9 activates the process so that the process can process the signal.
- If a signal trap is set up, F\$Send executes the signal processing routine (Intercept). If none was set up, the signal terminates the destination process, and the signal code becomes the exit status. (See the Wait system call.) An exception is the wakeup signal; that signal does not cause the signal intercept routine to be executed.
- Signal codes are defined as follows:
 - 0 = System terminate
(cannot be intercepted)
 - 1 = Wake up the process
 - 2 = Keyboard terminate
 - 3 = Keyboard interrupt
 - 128-255 = User defined
- If you try to send a signal to a process that has a signal pending, OS-9 cancels the current Send call, and returns an error. Issue a Sleep call for a few ticks; then, try again.
- The Sleep call saves CPU time. See the Intercept, Wait, and Sleep system calls for more information.

Sleep

Temporarily turns off
the calling process

OS9 F\$Sleep 103F 0A

Entry Conditions:

- X = One of the following:
 sleep time (in ticks)
 0 (sleep indefinitely)
 1 (sleep for the remainder of
 the current time slice)

Exit Conditions:

- X = *sleep time minus the number of ticks that the process was asleep*

Error Output:

- CC = carry set on error
B = *error code* (if any)

Additional Information:

- If Register X contains 0, OS-9 turns the process off until it receives a signal. Putting a process to sleep is a good way to wait for a signal or interrupt without wasting CPU time.
- If Register X contains 1, OS-9 turns the process off for the remainder of the process's current time slice. It inserts the process into the active process queue immediately. The process resumes execution when it reaches the front of the queue.
- If Register X contains an integer in the range 2-255, OS-9 turns off the process for the specified number of ticks, *n*. It inserts the process into the active process queue after *n-1* ticks. The process resumes execution when it reaches the front of the queue. If the process receives a signal, it awakens before the time has elapsed.
- When you select processes among multiple windows, you might need to set sleep for two ticks.

Set Priority

OS9 F\$SPrior 103F 0D

Changes the priority
of a process

Entry Conditions:

A = *process ID*
B = *priority*
 0 = lowest
 255 = highest

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- Set Priority changes the process's priority to the priority specified. A process can change another process's priority only if it has the same user ID.

Set SWI

OS9 F\$SSWI 103F 0E

Sets the SWI2 and
SWI3 vectors

Entry Conditions:

- A = SWI type code
- X = address of the user software interrupt routine

Exit Conditions:

- CC = carry set on error
- B = error code (if any)

Additional Information:

- Sets the interrupt vectors for SWI, SWI2 and SWI3 instructions.
- Each process has its own local vectors. Each Set SWI call sets one type of vector according to the code number passed in Register A:
 - 1 = SWI
 - 2 = SWI2
 - 3 = SWI3
- When OS-9 creates a process, it initializes all three vectors with the address of the OS-9 service call processor.
- **Warning:** Microware-supplied software uses SWI2 to call OS-9. If you reset this vector, these programs cannot work. If you change all three vectors, you cannot call OS-9 at all.

Set Time

Sets the system time and date

OS9 F\$STime 103F 16

Entry Conditions:

X = relative address of the time packet

Error Output:

CC = C bit set
B = error code

Additional Information:

- Set Time sets the current system date and time and starts the system real-time clock. The date and time are passed in a time packet as follows.

Relative Address	Value
0	year
1	month
2	day
3	hours
4	minutes
5	seconds

Then, the call makes a link system call to find the clock. If the link is successful, OS-9 calls the clock initialization. The clock initialization:

1. Sets up hardware dependent functions
2. Sets up the F\$STime system call via F\$SSvc

Set User ID Number

F\$SUser 103F 1C

Changes the current user ID without checking for errors or checking the ID number of the caller

Entry Conditions:

Y = *desired user ID number*

Error Output:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- The support module for this call is OS9p1.

Time

Gets the system date and time

OS9 F\$Time 103F 15

Entry Conditions:

X = address of the area in which to store the date and time packet

Exit Conditions:

X = date and time

Error Output:

CC = carry set on error
B = error code (if any)

Additional Information:

- The Time call returns the current system date and time in the form of a 6-byte packet (in binary). OS-9 copies the packet to the address passed in Register X.
- The packet looks like this:

Relative Address	Value
0	year
1	month
2	day
3	hours
4	minutes
5	seconds

- Time is a part of the clock module and it does not exist if no previous call to F\$Time has been made.

Unlink

OS9 F\$UnLink 103F 02

Unlinks (removes from memory) a module that is not in use and that has a link count of 0

Entry Conditions:

U = *address of the module header*

Error Output:

CC = *carry set on error*

B = *error code (if any)*

Additional Information:

- Unlink unlinks a module from the current process's address space, decreases its link count by one and, if the link count becomes zero, returns the memory where the module was located to the system for use by other processes.
- You cannot unlink system modules or device drivers that are in use.
- Unlink operates in the following order:
 1. Unlink tells OS-9 that the calling process no longer needs the module.
 2. OS-9 decreases the module's link count by one.
 3. When the resulting link count is zero, OS-9 destroys the module.

If any other process is using the module, the module's link count cannot fall to zero. Therefore, OS-9 does not destroy the module.

- If you pass a bad address, Unlink cannot find a module in the module directory and does not return an error.

Unlink A Module By Name

F\$UnLoad 103F 1D

Decrements a specified module's link count, and removes the module from memory if the resulting link count is zero

Entry Conditions:

A = *module type*
X = *pointer to module name*

Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- This system call differs from Unlink in that it uses a pointer to the module name, instead of the address of the module header.
- The support module for this call is OS9p2.

Wait

Temporarily turns off a calling process

OS9 F\$Wait 103F 04

Entry Conditions: None

Exit Conditions:

- A = *deceased child process's ID*
- B = *deceased child process's exit status code* (if no error)

Error Output:

- CC = *carry set on error*
- B = *error code* if any

Additional Information:

- The Wait call turns off the calling process until a child process *dies*, either by executing an Exit system call, or by receiving a signal. The Wait call helps you save system time.
- OS-9 returns the child's process's ID and exit status to the parent. If the child died because of a signal, the exit status byte (Register B) contains the signal code.
- If the caller has several children, OS-9 activates the caller when the first one dies. Therefore, you need to use one Wait system call to detect the termination of each child.
- OS-9 immediately reactivates the caller if a child dies before the Wait call. If the caller has no children, Wait returns an error. (See the Exit system call for more information.)
- If the Wait call returns with the carry bit set, the Wait function was not successful. If the carry bit is cleared, Wait functioned normally and any error that occurred in the child process is returned in Register B.

I/O User System Calls

Attach

OS9 I\$Attach 103F 80

Attaches a device to the system or verifies device attachment

Entry Conditions:

- A = *access mode*
- X = *address of the device name string*

Exit Conditions:

- X = *updated past device name*
- U = *address of the device table entry*

Error Output:

- B = *error code (if any)*
- CC = *carry set on error*

Additional Information:

- Attach does not reserve the device. It only prepares the device for later use by any process.
- OS-9 installs most devices automatically on startup. Therefore, you need to use Attach only when installing a device dynamically or when verifying the existence of a device. You need not use the Attach system call to perform routine I/O.
- The access mode parameter specifies the read and/or write operations to be allowed. These are:
 - 0 = Use any special device capabilities
 - 1 = Read only
 - 2 = Write only
 - 3 = Update (read and write)

- Attach operates in this sequence:
 1. OS-9 searches the system module to see if memory contains a device descriptor that has the same name as the device.
 - 2a. OS-9's next operation depends on whether or not the device is already attached. If OS-9 finds the descriptor and if the device is not already attached, OS-9 links the device's file manager and device driver. It then places the address of the manager and the driver in a new device table entry. OS-9 then allocates any memory needed by the device driver, and calls the driver's initialization routine which initializes the hardware.
 - b. If OS-9 finds the descriptor, and if the device is already attached, OS-9 verifies the attachment.

Change Directory

OS9 I\$Chgdir 103F 86

Changes the working directory of a process to a directory specified by a pathlist

Entry Conditions:

- A = *access mode*
- X = *address of the pathlist*

Exit Conditions:

- X = *updated past pathlist*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- If the access mode is read, write, or update, OS-9 changes the current data directory. If the access mode is execute, OS-9 changes the current execution directory.
- The calling process must have read access to the directory specified (public read if the directory is not owned by the calling process).
- The access modes are:
 - 1 = Read
 - 2 = Write
 - 3 = Update (read and write)
 - 4 = Execute

Close Path

Terminates an I/O path

OS9 I\$Close 103F 8F

Entry Conditions:

A = *path number*

Error Output:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- Close Path terminates the I/O path to the file or device specified by *path number*. Until you use another Open, Dup, or Create system call for that path, you can no longer perform I/O to the file or device.
- If you close a path to a single-user device, the device becomes available to other requesting processes. OS-9 de-allocates internally managed buffers and descriptors.
- The Exit system call automatically closes all open paths. Therefore, you might not need to use the Close Path system call to close some paths.
- Do not close a standard I/O path unless you want to change the file or device to which it corresponds.
- Close Path performs an implied I\$Detach call. If it causes the device link count to become 0, the device termination routine is executed. See I\$Detach for additional information.

Create File

Creates and opens a
disk file

OS9 I\$Create 103F 83

Entry Conditions:

- A = *access mode* (write or update)
- B = *file attributes*
- X = *address of the pathlist*; (See the following example.)

Exit Conditions:

- A = *path number*
- X = *address of the last byte of the pathlist + 1*; skips any trailing blanks (See the following example.)

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- OS-9 parses the pathlist and enters the new filename in the specified directory. If you do not specify a directory, OS-9 enters the new filename in the the working directory.
- OS-9 gives the file the attributes passed in Register B, which has bits defined as follows:

Bit	Definition
0	Read
1	Write
2	Execute
3	Public read
4	Public write
5	Public execute
6	Shareable file

- The access mode parameter passed in Register A must have the write bit set if any data is to be written. These access codes are defined as follows: 2 = write; 3 = update. The mode affects the file only until the file is closed.

- You can reopen the file in any access mode allowed by the file attributes. (See the Open system call.)
- Files opened for write can allow faster data transfer than those opened for update because update sometimes needs to pre-read sectors.
- If the execute bit (Bit 2) is set, the file is created in the working execution directory instead of the working data directory.
- Create File causes an implicit I\$Attach call. If the device has not previously been attached, the device's initialization routine is called.
- Later I/O calls use the path number to identify the file, until the file is closed.
- OS-9 does not allocate data storage for a file at creation. Instead, it allocates the storage either automatically when you first issue a write or explicitly by the Setstat subroutine.
- If the filename already exists in the directory, an error occurs. If the call specifies a non-multiple file device (such as a printer or terminal), Create behaves the same as Open.
- You cannot use Create to make directories. (See the Make Directory system call for instructions on how to do make directories.)
- Before the Create File call:

/	D	0	/	W	O	R	K	\$0D
---	---	---	---	---	---	---	---	------

↑
X

After the Create File call:

/	D	0	/	W	O	R	K	\$0D
---	---	---	---	---	---	---	---	------

↑
X

Delete File

Deletes a specified disk file

OS9 I\$Delete 103F 87

Entry Conditions:

X = *address of the pathlist* (See the following example.)

Exit Conditions:

X = *address of the last byte of the pathlist + 1*; any trailing blanks are skipped (See the following example.)

Error Output:

B = *error code* (if any)

CC = carry set on error

Additional Information:

- The Delete File call deletes the disk file specified by the pathlist. The file must have write permission attributes (public write, if the calling process is not the owner). An attempt to delete a device results in an error. The caller must have non-shareable write access to the file or an error results.

Example:

Before the Delete File call:

/	D	0	/	W	O	R	K	b	b	b	M	E	M	O	\$0D
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

↑
X

After the Delete File call:

/	D	0	/	W	O	R	K	b	b	b	M	E	M	O	\$0D
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

↑
X

Delete A File

OS9 I\$DeletX 103F 90

Deletes a file from the current data or current execution directory

Entry Conditions:

A = *access mode*
X = *address of the pathlist*

Exit Conditions:

X = *address of the last byte of the pathlist + 1*; any trailing blanks are skipped

Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- The Delete A File call removes the disk file specified by the selected pathlist. This function is similar to I\$Delete except that it accepts an access mode byte. If the access mode is execute, the call selects the current execution directory. Otherwise, it selects the current data directory.
- If a complete pathlist is provided (the pathlist begins with a slash (/), the access mode the call ignores the access mode.
- Only use this call to delete a file. If you attempt to use I\$DeletX to delete a device, the system returns an error.

Detach Device

OS9 I\$Detach 103F 81

Removes a device
from the system
device table

Entry Conditions:

U = *address of the device table entry*

Exit Conditions:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- The Detach Device call removes a device from both the system and the system device table, assuming the device is not being used by another process. You must use this call to detach devices attached using the Attach system call. Attach and Detach are both used mainly by the IO manager. SCF also uses Attach and Detach to set up its second device (echo device).
- This is the sequence of the operation of Detach Device:
 1. Detach Device calls the device driver's termination routine. Then, OS-9 deallocates any memory assigned to the driver.
 2. OS-9 unlinks the associated device driver and file manager modules.
 3. OS-9 then removes the driver, as long as no other module is using that driver.

Duplicate Path

OS9 I\$Dup 103F 82

Returns a synonymous
path number

Entry Conditions:

A = *old path number* (number of path to duplicate)

Exit Conditions:

A = *new path number* (if no error)

Error Output:

B = *error code* (if error encountered)

CC = carry set on error

Additional Information:

- The Duplicate Path returns another, synonymous path number for the file or device specified by the *old path number*.
- The shell uses the Duplicate Path call when it redirects I/O.
- System calls can use either path number (old or new) to operate on the same file or device.
- Make sure that no more than one process is performing I/O on any one path at the same time. Concurrent I/O on the same path can cause unpredictable results with RBF files.
- The I\$Dup call always uses the lowest available path number. This lets you manipulate standard I/O paths to contain any desired paths.

Get Status

Returns the status of a file or device

OS9 I\$GetStt 103F 8D

Entry Conditions:

A = *path number*
B = *function code*

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- The Status is a *wildcard* call. Use it to handle device parameters that:
 - Are not the same for all devices
 - Are highly hardware-dependent
 - Must be user-changeable
- The exact operation of the Get Status system call depends on the device driver and file manager associated with the path. A typical use is to determine a terminal's parameters for such functions as backspace character and echo on/off. The Get Status call is commonly used with the Set Status call.
- The Get Status function codes that are currently defined are listed in the "Get Status System Calls" section.

Make Directory

Creates and initializes
a directory

OS9 I\$MakDir 103F 85

Entry Conditions:

B = *directory attributes*
X = *address of the pathlist*

Exit Conditions:

X = *address of the last byte of the pathlist + 1*; Make Directory skips trailing blanks.

Error Output:

B = *error code (if any)*
CC = *carry set on error*

Additional Information:

- The Make Directory call creates and initializes a directory as specified by the pathlist. The directory contains only two entries, one for itself (.) and one for its parent directory (..)
- OS-9 makes the calling process the owner of the directory.
- Because the Make Directory call does not open the directory, it does not return a path number.
- The new directory automatically has its directory bit set in the access permission attributes. The remaining attributes are specified by the byte passed in Register B. The bits are defined as follows:

Bit	Definition
0	Read
1	Write
2	Execute
3	Public read
4	Public write
5	Public execute
6	Single-user
7	Don't care

- Before the Make Directory call:

/	D	0	/	N	E	W	D	I	R	\$0D
---	---	---	---	---	---	---	---	---	---	------

↑
X

After the Make Directory call:

/	D	0	/	N	E	W	D	I	R	\$0D
---	---	---	---	---	---	---	---	---	---	------

↑
X

Open Path

OS9 I\$Open 103F 84

Opens a path to an existing file or device as specified by the pathlist

Entry Conditions:

- A = *access mode* (D S PE PW PR E W R)
- X = *address of the pathlist* (See the following example.)

Exit Conditions:

- A = *path number*
- X = *address of the last byte of the pathlist + 1*

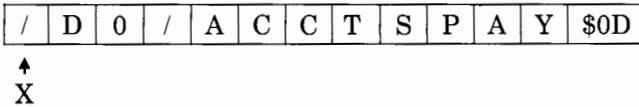
Error Output:

- B = *error code* (if any)
- CC = *carry set on error*

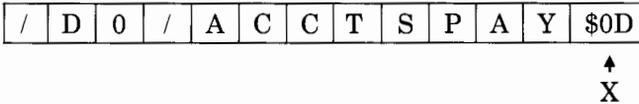
Additional Information:

- OS-9 searches for the file in one of the following:
 - The directory specified by the pathlist if the pathlist begins with a slash.
 - The working data directory, if the pathlist does not begin with a slash.
 - The working execution directory, if the pathlist does not begin with a slash and if the execution bit is set in the access mode.
- OS-9 returns a path number for later system calls to use to identify the file.
- The access mode parameter lets you specify which read and/or write operations are to be permitted. When set, each access mode bit enables one of the following: Write, Read, Read and Write, Update, Directory I/O.
- The access mode must conform to the access permission attributes associated with the file or device. (See the Create system call.) Only the owner can access a file unless the appropriate public permission bits are set.

- The update mode might be slightly slower than the others because it might require pre-reading of sectors for random access of bytes within sectors.
- Several processes (users) can open files at the same time. Each device has an attribute that specifies whether or not it is shareable.
- Before the Open Path call:



After the Open Path call:



- If the single-user bit is set, the file is opened for single-user access regardless of the settings of the file's permission bits.
- You must open directory files for read or write if the directory bit (Bit 7) is set in the access mode.
- Open Path always uses the lowest path number available for the process.

Read

Reads *n* bytes from a specified path

OS9 I\$Read 103F 89

Entry Conditions:

- A = *path number*
- Y = *number of bytes to read*
- X = *address in which to store the data*

Exit Conditions:

- Y = *number of bytes read*

Error Output:

- B = *error code (if any)*
- CC = *carry set on error*

Additional Information:

- The Read call reads the specified number of bytes from the specified path. It returns the data exactly as read from the file/device, without additional processing or editing. The path must be opened in the read or update mode.
- If there is not enough data in the specified file to satisfy the read request, the call reads fewer bytes than requested but an end-of-file error is **not** returned. After all data in a file is read, the next I\$Read call returns an end-of-file error.
- If the specified file is open for update, the record read is locked out on RBF-type devices.
- The keyboard terminate, keyboard interrupt, and end-of-file characters are filtered out of the Entry Conditions data on SCF-type devices unless the corresponding entries in the path descriptor have been set to zero. You might want to modify the device descriptor so that these values are initialized to zero when the path is opened.

- The call reads the number of bytes requested unless Read encounters any of the following:
 - An end-of-file character
 - An end-of-record character (SCF only)
 - An error

Read Line With Editing

Reads a text line with editing

OS9 I\$ReadLn 103F 8B

Entry Conditions:

A = *path number*
X = *address at which to store data*
Y = *maximum number of bytes to read*

Exit Conditions:

Y = *number of bytes read*

Error Output:

B = *error code (if any)*
CC = *carry set on error*

Additional Information:

- Read Line is similar to Read. The difference is that Read Line reads the input file or device until it encounters a carriage return character or until it reaches the maximum byte count specified, whichever comes first. The Read Line also automatically activates line editing on character oriented devices, such as terminals and printers. The line editing refers to auto line feed, null padding at the end of the line, backspacing, line deleting, and so on.
- SCF requires that the last byte entered be an end-of-record character (usually a carriage return). If more data is entered than the maximum specified, Read Line does not accept it and a PD.OVF character (usually a bell) is echoed.
- After one Read Line call reads all data in a file, the next Read Line call generates an end-of-file error.
- (For more information about line editing, see “SCF Line Editing Functions” in Chapter 6.)

Seek

Repositions a file
pointer

OS9 I\$Seek 103F 88

Entry Conditions:

- A = *path number*
- X = *MS 16 bits of the desired file position*
- U = *LS 16 bits of the desired file position*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- The Seek Call repositions the path's logical file pointer, the 32-bit address of the next byte in the file to be read from or written to.
- You can perform a seek to any value, regardless of the file's size. Later writes automatically expand the file to the required size (if possible). Later reads, however, return an end-of-file condition. Note that a seek to Address 0 is the same as a rewind operation.
- OS-9 usually ignores seeks to non-random access devices, and returns without error.
- On RBF devices, seeking to a new disk sector causes the internal disk buffer to be rewritten to disk if it has been modified. Seek does not change the state of record locking.

Set Status

Sets the status of a file or device

OS9 I\$SetStt 103F 8E

Entry Conditions:

A = *path number*

B = *function code*

Other registers depend on the function code.

Error Output:

B = *error code* (if any)

CC = carry set on error

Other registers depend on the function code.

Additional Information:

- Set Status is a wildcard call. Use it to handle device parameters that:
 - Are not the same for all devices
 - Are highly hardware-dependent
 - Must be user-changeable
- The exact operation of the Set Status system call depends on the device driver and file manager associated with the path. A typical use is to set a terminal's parameters for such functions as backspace character and echo on/off. The Set Status call is commonly used with the Get Status call.
- The Set Status function codes that are currently defined are listed in the "Set Status System Calls" section.

Write

Writes to a file or device

OS9 I\$Write 103F 8A

Entry Conditions:

- A = *path number*
- X = *starting address of data to write*
- Y = *number of bytes to write*

Exit Conditions:

- Y = *number of bytes written*

Error Output:

- B = *error code (if any)*
- CC = *carry set on error*

Additional Information:

- The Write system call writes to the file or device associated with the path number specified.
- Before using Write, be sure the path is opened or created in the Write or Update access mode. OS-9 writes data to the file or device without processing or editing the data. OS-9 automatically expands the file if you write data past the present end-of-file.

Write Line

OS9 I\$WritLn 103F 8C

Writes to a file or device until it encounters a carriage return

Entry Conditions:

A = *path number*
X = *address of the data to write*
Y = *maximum number of bytes to write*

Exit Conditions:

Y = *number of bytes written*

Error Output:

B = *error code (if any)*
CC = *carry set on error*

Additional Information:

- Writes to the file or device that is associated with the path number specified.
- Write Line is similar to Write. The difference is that Write Line writes data until it encounters a carriage return character. It also activates line editing for character-oriented devices, such as terminals and printers. The line editing refers to auto line feed, null padding at the end of the line, backspacing, line deleting, and so on.
- Before using Write Line, be sure the path is opened or created in the write or update access mode.
- (For more information about line editing, see “SCF Line Editing Functions” in Chapter 6.)

Privileged System Mode Calls

Set an alarm

OS9 F\$Alarm 103F 1E

Sets an alarm to ring the bell at a specified time

Entry Conditions:

X = *relative address of time packet*

Error Output:

CC = carry set on error

B = *appropriate error code*

Additional Information:

- When the system reaches the specified alarm time, it rings the bell for 15 seconds.
- The time packet is identical to the packet used in the F\$STime call. See F\$STime for additional information on the format of the packet.
- All alarms begin at the start of a minute and any seconds in the packet are ignored.
- The system is limited to one alarm at a time.

Allocate 64

OS9 F\$All64 103F 30

Dynamically allocates
64-byte blocks of
memory

Entry Conditions:

X = *base address of the page table*; 0 = the page table has not been allocated

Exit Conditions:

A = *block number*
X = *base address of the page table*
Y = *address of the block*

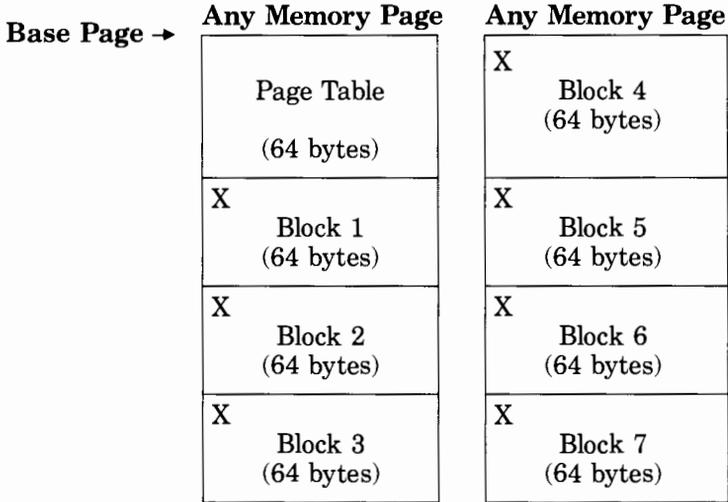
Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- The Allocate 64 system call allocates the 64-byte blocks of memory by splitting pages (256-byte sections) into four sections.
- OS-9 uses the first 64 bytes of the base page as a page table. This table contains the page number (most significant byte of the address) of all pages in the memory structure. If Register X passes a value of zero, the call allocates a new base page and the first 64-byte memory block.
- Whenever a new page is needed, a Request System Memory system call (F\$SRqMem) executes automatically.
- The first byte of each block contains the block number. Routines that use the Allocate 64 call should not alter this byte.

- The following diagram shows how seven blocks might be allocated:



Allocate High RAM

**Allocate system
memory from high
physical memory**

OS9 F\$AlHRam 103F 53

Entry Conditions:

B = *number of blocks*

Error Output:

CC = *carry set on error*

B = *appropriate error code*

Additional Information:

- This call searches for the desired number of contiguous free RAM blocks, starting its search at the top of memory. F\$AllHRam is similar to F\$AllRAM except F\$AllRAM begins its search at the bottom of memory.
- Screen allocation routines use this call to provide a better chance of finding the necessary memory for a screen.

Allocate Image

OS9 F\$AllImg 103F 3A

Allocates RAM
blocks for process
DAT image

Entry Conditions:

- A = *starting block number*
- B = *number of blocks*
- X = *process descriptor pointer*

Exit Conditions:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- Use the Allocate Image system call to allocate a data area for a process. The blocks that Allocate Image defines might not be contiguous.
- The support module for this system call is OS9p1.

Allocate Process Descriptor

Allocates and initializes a 512-byte process descriptor

OS9 F\$AllProc 103F 4B

Entry Conditions: None

Exit Conditions:

U = *process descriptor pointer*

Error Output:

CC = C bit set on error

B = *appropriate error code*

Additional Information:

- The process descriptor table houses the address of the descriptor. Initialization of the process descriptor consists of clearing the first 256 bytes of the descriptor, setting up the state as a system state, and marking as unallocated as much of the DAT image as the system allows—typically, 60-64 kilobytes.
- The support module for this system call is OS9p2. The call also branches to the F\$SRqMem call.

Allocate RAM

OS9 F\$AllRAM 103F 39

Searches the memory block map for the desired number of contiguous free RAM blocks

Entry Conditions:

B = *number of blocks*

Exit Conditions:

CC = C bit set on error

B = *appropriate error code*

Additional Information:

- The support module for this system call is OS9p1.

Allocate Process Task Number

OS9 F\$AllTsk 103F 3F

Determines whether OS-9 has assigned a task number to the specified process

Entry Conditions:

X = *process descriptor pointer*

Error Output:

CC = C bit set

B = *appropriate error code*

Additional Information:

- If the process does not have a task number, OS-9 allocates a task number and copies the DAT image into the DAT hardware.
- The support module for this call is OS9p1. Allocate Process Task number also branches to F\$ResTsk and F\$SetTsk.

Insert Process

OS9 F\$AProc 103F 2C

Inserts a process into
the queue for execution

Entry Conditions:

X = *address of the process descriptor*

Error Output:

CC = *carry set on error*

B = *error code (if any)*

Additional Information:

- The Insert Process system call inserts a process into the active process queue so that OS-9 can schedule the process for execution.
- OS-9 sorts all processes in the queue by process age (the count of how many process switches have occurred since the process's last time slice). When a process is moved to the active process queue, OS-9 sets its age according to its priority—the higher the priority, the higher the age.

An exception is a newly active process that was deactivated while in the system state. OS-9 gives such a process higher priority because the process usually is executing critical routines that affect shared system resources.

Bootstrap System

OS9 F\$Boot 103F 35

Links either the module named Boot or the module specified in the INIT module

Entry Conditions: None

Error Output:

CC = *C bit set on error*
B = *appropriate error code*

Additional Information:

- When it calls the linked module, Boot expects to receive a pointer giving it the location and size of an area in which to search for the new module.
- The support module for this call is OS9p1. Bootstrap System also branches to the F\$Link and F\$VModul system calls.

Bootstrap Memory Request

OS9 F\$BtMem 103F 36

Allocates the requested memory (rounded to the nearest block) as contiguous memory in the system's address space

Entry Conditions:

D = *byte count requested*

Exit Conditions:

D = *byte count granted*

U = *pointer to memory allocated*

Error Output:

CC = *C bit set on error*

B = *appropriate error code*

Additional Information:

- This call is identical to F\$SRqMem.
- The Bootstrap Memory Request support module is OS9p1.

Clear Specified Block

Marks blocks in the process DAT image as unallocated

OS9 F\$ClrBlk 103F 50

Entry Conditions:

- B = *number of blocks*
- U = *address of first block*

Exit Conditions: None

Additional Information:

- After Clear Specified Block deallocates blocks, the blocks are free for the process to use for other data or program areas. If the block address passed to Clear Specified Block is invalid or if the call attempts to clear the stack area, it returns E\$IBA.
- The support module for the call is OS9p2.

DAT to Logical Address

OS9 F\$DATLog 103F 44

Converts a DAT image clock number and block offset to its equivalent logical address

Entry Conditions:

B = DAT image offset
X = block offset

Exit Conditions:

X = logical address

Error Output:

CC = C bit set on error
B = appropriate error code

Additional Information:

- Following is a sample conversion:

2000 - 2FFF	Input: B = 2 X = \$0329
1000 - 1FFF	Output: X = \$2329
0 - FFF	

- The support module for this call is OS9p1.

Deallocate Image RAM Blocks

Deallocates image
RAM blocks

OS9 F\$DelImg 103F 3B

Entry Conditions:

- A = *number of starting block*
- B = *block count*
- X = *process descriptor pointer*

Error Output:

- CC = C bit set on error
- B = *appropriate error code*

Additional Information:

- This system call deallocates memory from a process's address space. It frees the RAM for system use and frees the DAT image for the process. Its main use is to let the system clean up after a process death.
- The support module for this call is OS9p2.

Deallocate Process Descriptor

Returns a process
descriptor's memory to
a free memory pool

OS9 F\$DelPrc 103F 4C

Entry Conditions:

A = *process ID*

Error Output:

CC = C bit set on error

B = *appropriate error code*

Additional Information:

- Use this call to clear the system scratch memory and stack area associated with the process.
- The support module for this call is OS9p2.

Deallocate RAM blocks

OS9 F\$DelRAM 103F 51

Clears a block's RAM In Use flag in the system memory block map

Entry Conditions:

- B = *number of blocks*
- X = *starting block number*

Exit Conditions: None

Additional Information:

- The Deallocate RAM Blocks call assumes the blocks being deallocated are not associated with any DAT image.
- The support module for this call is OS9p2.

Deallocate Task Number

OS9 F\$DelTsk 103F 40

Releases the task number that the process specified by the passed descriptor pointer

Entry Conditions:

X = *process descriptor pointer*

Error Output:

CC = C bit set on error

B = *appropriate error code*

Additional Information:

- The support module for this call is OS9p1.

Link Using Module Directory Entry

Performs a link using a
pointer to a module
directory entry

OS9 F\$ELink 103F 4D

Entry Conditions:

B = *module type*
X = *pointer to module directory entry*

Exit Conditions:

U = *module header address*
Y = *module entry point*

Error Output:

CC = *C bit set on error*
B = *appropriate error code*

Additional Information:

- This call differs from Link in that you supply a pointer to the module directory entry rather than a pointer to a module name.
- The support module for this call is OS9p1.

Find Module Directory Entry

Returns a pointer to the module directory entry

OS9 F\$FModul 103F 4E

Entry Conditions:

- A = *module type*
- X = *pointer to the name string*
- Y = *DAT image pointer (for name)*

Exit Conditions:

- A = *module type*
- B = *module revision number*
- X = *updated name string; (if Register A contains 0 on entry)*
- U = *module directory entry pointer*

Error Output:

- CC = *C bit set on error*
- B = *appropriate error code*

Additional Information:

- The Find Module Directory Entry call returns a pointer to the module directory entry for the first module that has a name and type matching the specified name and type. If you pass a module type of zero, the system call finds any module.
- The support module for this call is OS9p1.

Find 64

OS9 F\$Find64 103F 2F

Returns the address
of a 64-byte memory
block

Entry Conditions:

A = *block number*
X = *address of the block*

Exit Conditions:

Y = *address of the block*
CC = carry set if block not allowed or not in use

Additional Information:

- OS-9 uses Find 64 to find path descriptors when given their block number. The block number can be any positive integer.

Get Free High Block

OS9 F\$FreeHB 103F 3E

Searches the DAT image for the highest set of contiguous free blocks of the specified size

Entry Conditions:

B = *block count*
Y = *DAT image pointer*

Exit Conditions:

A = *starting block number*

Error Output:

CC = *C bit set on error*
B = *appropriate error code*

Additional Information:

- The Get Free High Block call returns the block number of the beginning memory address of the free blocks.
- The support module for this system call is OS9p1.

Get Free Low Block

OS9 F\$FreeLB 103F 3D

Searches the DAT image for the lowest set of contiguous free blocks of the specified size

Entry Conditions:

B = *block count*
Y = *DAT image pointer*

Exit Conditions:

A = *starting block number*

Error Output:

CC = *C bit set on error*
B = *appropriate error code*

Additional Information:

- The Get Free Low Block call returns the block number of the beginning memory address of the free blocks.
- The support module for this system call is OS9p1.

Compact Module Directory

Compacts the entries in the module directory

OS9 F\$GCMDir 103F 52

Entry Conditions: None

Exit Conditions: None

Additional Information:

- This function is only for internal OS-9 system use. You should never call it from a program.

Get Process Pointer

**Gets a pointer to a
process**

F\$GProcP 103F 37

Entry Conditions:

A = *process ID*

Exit Conditions:

B = *pointer to process descriptor* (if no error)

Error Output:

CC = *carry set on error*

B = *error code* (If an error occurs (E\$BPrCID))

Additional Information:

- The Get Process Pointer call translates a process ID number to the address of its process descriptor in the system address space. Process descriptors exist only in the system task address space. Because of this, the address returned only refers to system address space.
- The support module for this call is OS9p2.

I/O Delete

Deletes an I/O module
that is not being used

OS9 F\$IODel 103F 33

Entry Conditions:

X = *address of an I/O module*

Error Output:

CC = *carry set on error*

B = *error code (if any)*

Additional Information:

- The I/O Delete deletes the specified I/O module from the system, if the module is not in use. This system call is used mainly by the I/O MANAGER, and can be of limited or no use for other applications.
- This is the order in which I/O Delete operates:
 1. Register X passes the address of a device descriptor module, device driver module, or file manager module.
 2. OS-9 searches the device table for the address.
 3. If OS-9 finds the address, it checks the module's use count. If the count is zero, the module is not being used; OS-9 deletes it. If the count is not zero, the module is being used; OS-9 returns an error.
- I/O Delete returns information to the Unlink system call after determining whether a device is busy.

I/O Queue

OS9 F\$IOQu 103F 2B

Inserts the calling process into another process's I/O queue, and puts the calling process to sleep

Entry Conditions:

A = *process number*

Error Output:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- The I/O Queue call links the calling process into the I/O queue of the specified process and performs an untimed sleep. The IO Manager and the file managers are primary and extensive users of I/O Queue.
- Routines associated with the specified process send a wake-up signal to the calling process.

Set IRQ

OS9 F\$IRQ 103F 2A

Adds a device to or removes it from the polling table

Entry Conditions:

- D = *address of the device status register*
X = 0 (to remove a device) or *the address of a packet* (to add a device)
- the address at X is the flip byte
 - the address at X + 1 is the mask byte
 - the address at X + 2 is the priority byte
- Y = *address of the device IRQ service routine*
U = *address of the service routine's memory area*

Error Output:

- CC = *carry set on error*
B = *error code* (if any)

Additional Information:

- Set IRQ is used mainly by device driver routines. (See "Interrupt Processing" in Chapter 2 for a complete discussion of the interrupt polling system.)
- Packet Definitions:

The Flip Byte. Determines whether the bits in the device status register indicate active when set or active when cleared. If a bit in the flip byte is set, it indicates that the task is active whenever the corresponding bit in the status register is clear (and vice versa).

The Mask Byte. Selects one or more bits within the device status register that are interrupt request flag(s). One or more set bits identify which task or device is active.

The Priority Byte. Contains the device priority number (0 = lowest priority, 255 = highest priority).

Load A From Task B

Loads A from 0,X in task B

F\$LDABX 103F 49

Entry Conditions:

B = *task number*
X = *pointer to data*

Exit Conditions:

A = *byte at 0,X in task address space*

Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- The value in Register X is an offset value from the beginning address of the Task module. The Load A From Task B call returns one byte from this logical address. Use this system call to get one byte from the current process's memory in a system state routine.

Get One Byte

Loads A from [X, [Y]]

F\$LDAXY 103F 46

Entry Conditions:

X = *block offset*
Y = *DAT image pointer*

Exit Conditions:

A = *contents of byte at DAT image (Y) offset by X*

Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- The Get One Byte system call gets the contents of one byte in the specified memory block. The block is specified by the DAT image in (Y), offset by (X). The call assumes that the DAT image pointer is to the actual block desired, and that X is only an offset within the DAT block. The value in Register X must be less than the size of the DAT block. OS-9 does not check to see if X is out of range.

Get Two Bytes

F\$LDDDX 103F 48

Loads D from
[D + X],[Y]

Entry Conditions:

- D = *Offset to the offset within the DAT image*
- X = *Offset within the DAT image*
- Y = *DAT image pointer*

Exit Conditions:

- D = *contents of two bytes at [D + X,Y]*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- Get Two Bytes loads two bytes from the address space described by the DAT image pointer. If the DAT image pointer is to the entire DAT, then make D + X equal to the process address for data. If the DAT image is not the entire image (64K), then you must adjust D + X relative to the beginning of the DAT image. Using D + X lets you keep a local pointer within a block, and also lets you point to an offset within the DAT image that points to a specified block number.

Map Specific Block

F\$MapBlk 103F 4F

Maps the specified block(s) into unallocated blocks of process space

Entry Conditions:

X = *starting block number*
B = *number of blocks*

Exit Conditions:

U = *address of first block*

Error Output:

B = *error code (if any)*
CC = *carry set on error*

Additional Information:

- The system maps blocks from the top down. It maps new blocks into the highest available addresses in the address space. See Clear Specified Block for information on unmapping.

Move Data

F\$Move 103F 38

Moves data bytes from one address space to another

Entry Conditions:

- A = *source task number*
- B = *destination task number*
- X = *source pointer*
- Y = *byte count*
- U = *destination pointer*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- You can use the Move Data system call to move data bytes from one address space to another, usually from system to user, or vice versa.
- The support module for this call is OS9p1.

Next Process

OS9 F\$NProc 103F 2D

**Executes the next
process in the active
process queue**

Entry Conditions: None

Exit Conditions:

Control does not return to caller.

Additional Information:

- The Next Process system call takes the next process out of the active process queue and initiates its execution. If the queue contains no process, OS-9 waits for an interrupt, and then checks the queue again.
- The process calling Next Process must already be in one of the three process queues. If it is not, it becomes unknown to the system even though the process descriptor still exists and can be displayed by a PROCS command.

Release A Task

F\$RelTsk 103F 43

Releases a specified DAT task number from use by a process, making the task's DAT hardware available for use by another task

Entry Conditions:

B = *task number*

Error Output:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- The support module for this call OS9p1.

Reserve Task Number

Reserves a DAT task number

F\$ResTsk 103F 42

Entry Conditions: none

Exit Conditions:

B = *task number* (if no error)

Error Output:

CC = carry set on error

B = *error code* if an error occurs

Additional Information:

- The Reserve Task Number call finds a free DAT task number, reserves it, and returns the task number to the caller. The caller often then assigns the task number to a process.
- The support module for this call is OS9p1.

Return 64

OS9 F\$Ret64 103F 31

**Deallocates a 64-byte
block of memory**

Entry Conditions:

A = *block number*
X = *address of the base page*

Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- See the Allocate 64 system call for more information.

Set Process DAT Image

Copies all or part of the DAT image into a process descriptor

F\$SetImg 103F 3C

Entry Condition:

- A = *starting image block number*
- B = *block count*
- X = *process descriptor pointer*
- U = *new image pointer*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- While copying part or all of the DAT image, this system call also sets an image change flag in the process descriptor. This flag guarantees that as a process returns from the system call. The call updates the hardware to match the new process DAT image.
- The support module for this call is OS9p1.

Set Process Task DAT Registers Writes to the hardware DAT registers

F\$SetTsk 103F 41

Entry Conditions:

X = *pointer to process descriptor*

Error Output:

CC = carry set on error

B = *error code* (if any)

Additional Information:

- This system call sets the process task hardware DAT registers, and clears the image change flag in the process descriptor. It also writes to DAT RAM the process's segment address information.
- The support module for this call is OS9p1.

System Link

F\$SLink 103F 34

Adds a module from outside the current address space into the current address space

Entry Conditions:

- A = *module type*
- X = *module name string pointer*
- Y = *name string DAT image pointer*

Exit Conditions:

- A = *module type*
- B = *module revision (if no error)*
- X = *updated name string pointer*
- Y = *module entry point*
- U = *module pointer*

Error Output:

- CC = *carry set on error*
- B = *error code (If an error occurs)*

Additional Information:

- The I/O System uses the System Link call to link into the current process's address space those modules specified by a device name in a user call. User calls such as Create File and Open Path use this System Link.
- The support module for this call is OS9p1.

Request System Memory

Allocates a block of memory of the specified size from the top of available RAM

OS9 F\$SRqMem 103F 28

Entry Conditions:

D = *byte count*

Exit Conditions:

U = *starting address of the memory area*

D = *new memory size*

Error Output:

CC = *carry set on error*

B = *error code (if any)*

Additional Information:

- The Request System Memory call rounds the size request to the next page boundary.
- This call allocates memory only for system address space.

Return System Memory

Deallocates a block of contiguous pages

OS9 F\$SRtMem 103F 29

Entry Conditions:

- U = *starting address of memory to return*; must point to an even page boundary
- D = *number of bytes to return*

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- Register U must point to an even page boundary.
- This call deallocates memory for system address space only.

Set SVC

OS9 F\$SSvc 103F 32

Adds or replaces a system call

Entry Conditions:

Y = address of the system call initialization table

Error Output:

CC = C bit set
 B = error code

Additional Information:

- Set SVC adds or replaces a system call, which you have written, to OS-9's user and system mode system call tables.
- Register Y passes the address of a table, which contains the function codes and offsets, to the corresponding system call handler routines. This table has the following format:

Relative Use
 Address

\$00	Function Code	← First entry
\$01	__ Offset From Byte 3 __	
\$02	To Function Handler	
\$03	Function Code	← Second entry
\$04	__ Offset From Byte 6 __	
\$05	To Function Handler	
	More Entries	← More entries
	\$80	← End-of-table mark

- If the most significant bit of the function code is set, OS-9 updates the system table.

If the most significant bit of the function code is not set, OS-9 updates the system and user tables.

- The function request codes are in the range \$29-\$34. IO calls are in the range \$80-\$90
- To use a privileged system call, you must be executing a program that resides in the system map and that executes in the system state.
- The system call handler routine must process the system call and return from the subroutine with an RTS instruction.
- The handler routine might alter all CPU registers (except Register SP).
- Register U passes the address of the register stack to the system call handler as shown in the following diagram:

		Relative Address	Name
U →	CC	\$00	R\$CC
		\$01	R\$D
	A	\$01	R\$A
	B	\$02	R\$B
	DP	\$03	R\$DP
	X	\$04	R\$X
	Y	\$06	R\$Y
	U	\$08	R\$U
	PC	\$0A	R\$PC

Codes \$70-\$7F are reserved for user definition.

Store A Byte In A Task

Stores A at 0,X in
Task B

F\$STABX 103F 4A

Entry Conditions:

- A = *byte to store*
- B = *destination task number*
- X = *logical destination address*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- This system call is similar to the assembly language instruction “STA 0,X”. The difference is that in the system call, X refers to an address in the given task’s address space, instead of the current address space.
- The support module for this system call is OS9p1.

Install virtual interrupt

Installs a virtual interrupt handler routine

OS9 F\$VIRQ 103F 27

Entry Conditions:

D = *initial count value*
X = 0 to delete entry
 1 to install entry
Y = *address of 5-byte packet*

Error Output:

CC = carry set on error
B = *appropriate error code*

Additional Information:

- Install VIRQ for use with devices in the Multi-Pak Expansion Interface. This call is explained in detail in Chapter 2.

Validate Module

OS9 F\$VModul 103F 2E

Checks the module header parity and CRC bytes of a module

Entry Conditions:

D = *DAT image pointer*
X = *new module block offset*

Exit Conditions:

U = *address of the module directory entry*

Error Output:

CC = *carry set on error*
B = *error code (if any)*

Additional Information:

- If the values of the specified module are valid, OS-9 searches the module directory for a module with the same name. If one exists, OS-9 keeps in memory the module that has the higher revision level. If both modules have the same revision level, OS-9 retains the module in memory.

Get Status System Calls

You use the Get Status system calls with the RBF manager subroutine GETSTA. The OS-9 Level Two system reserves function Codes 7-127 for use by Microware. You can define Codes 128-255 and their parameter-passing conventions for your own use. (See the sections on device drivers in Chapters 4, 5, and 6.)

The Get Status routine passes the register stack and the specified function code to the device driver.

Following are the Get Status functions and their codes.

SS.OPT

(Function code \$00). Reads the option section of the path descriptor, and copies it into the 32-byte area pointed to by Register X

Entry Conditions:

A = *path number*
B = \$00
X = *address to receive status packet*

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- Use SS.OPT to determine the current settings for editing functions, such as echo and auto line feed.

SS.RDY

(Function code \$01). Tests for data available on SCF-supported devices

Entry Conditions:

A = *path number*
B = \$01

Exit Conditions:

If the device is ready:

CC = carry clear
B = \$00

If the device is not ready:

CC = carry set
B = \$F6 (E\$SRNDY)

Error Output:

CC = carry set
B = *error code*

SS.SIZ

(Function code \$02). Gets the current file size on a RBF-supported devices only

Entry Conditions:

A = *path number*
B = \$02

Exit Conditions:

X = *most significant 16 bits of the current file size*
U = *least significant 16 bits of the current file size*

Error Output:

CC = carry set on error
B = *error code* (if any)

SS.POS

(Function code \$05). Gets the current file position (RBF-supported devices only)

Entry Conditions:

A = *path number*
B = \$05

Exit Conditions:

X = *most significant 16 bits of the current file position*
U = *least significant 16 bits of the current file position*

Error Output:

CC = *carry set on error*
B = *error code (if any)*

SS.EOF

(Function code \$06). Tests for the end of the file (EOF)

Entry Conditions:

A = *path number*
B = \$06

Exit Conditions:

If there is no EOF:

CC = *carry clear*
B = \$00

If there is an EOF:

CC = *carry set*
B = \$D3 (E\$EOF)

Error Output:

CC = *carry set*
B = *error code*

SS.DevNm

(Function Code \$0E). Returns a device name

Entry Conditions:

- A = *path number*
- B = \$0E
- X = *address of 32-byte buffer for name*

Exit Conditions:

- X = *address of buffer, name moved to buffer*

SS.DSTAT

(Function code \$12). Returns the display status

Entry Conditions:

- A = *path number*
- B = \$12

Exit Conditions:

- A = *color code of the pixel at the cursor address*
- X = *address of the graphics display memory*
- Y = *graphics cursor address; X = MSB, Y = LSB*

Additional Information:

- This function is supported only with the VDGINT module and deals with VDG-compatible graphics screens. See SS.AAGBf for information regarding Level Two operation.

SS.JOY

(Function code \$13). Returns the joystick values

Entry Conditions:

- A = *path number*
- B = \$13
- X = *joystick number*
 - 0 = (right joystick)
 - 1 = (left joystick)

Exit Conditions:

- A = *fire button down*
 - 0 = none
 - 1 = Button 1
 - 2 = Button 2
 - 3 = Button 1 and Button 2
- X = *selected joystick x value (0-63)*
- Y = *selected joystick y value (0-63)*

Note: Under Level 1, the following values are returned by this call:

- A = fire button status
 - \$FF = fire button is on
 - \$00 = fire button is off

SS.AlfaS

(Function code \$1C). Returns VDG alpha screen memory information

Entry Conditions:

A = *path number*
B = \$1C

Exit Conditions:

A = *caps lock status*
 \$00 = lower case
 \$FF = upper case
X = *memory address of the buffer*
Y = *memory address of the cursor*

Additional Information:

- SS.AlfaS maps the screen into the user address space. The call requires a full block of memory for screen mapping. This call is only for use with VDG text screens handled by VDGINT.
- The support module for this call is VDGINT.
- **Warning:** Use extreme care when poking the screen, since other system variables reside in screen memory. Do not change any addresses outside of the screen.

SS.Cursr

(**Function code \$25**). Returns VDG alpha screen cursor information

Entry Conditions:

A = *path number*
B = \$25

Exit Conditions:

A = *character code of the character at the current cursor address*
X = *cursor X position (column)*
Y = *cursor Y position (row)*

Additional Information:

- SS.Cursr returns the character at the current cursor position. It also returns the X-Y address of the cursor relative to the current device's window or screen. SS.Cursr works only with text screens.
- The support module for this call is VDGINT.

SS.ScSiz

(Function code \$26). Returns the window or screen size

Entry Conditions:

A = *path number*
B = \$26

Exit Conditions:

X = *number of columns on screen/window*
Y = *number of rows on screen/window*

Additional Information:

- Use this call to determine the size of an output screen. The values returned depend on the device in use:
 - For non-CCIO devices, the call returns the values following the XON/XOFF bytes in the device descriptor.
 - For CCIO devices, the call returns the size of the window or screen in use by the specified device.
 - For window devices, the call returns the size of the current working area of the window.
- The support modules for this call are VDGINT, GrfInt, and WindInt.

SS.KySns

(Function code \$27). Returns key down status

Entry Conditions:

- A = path number
- B = \$27

Exit Conditions:

- A = keyboard scan information

Additional Information:

- Accumulator A returns with a bit pattern representing eight keys. With each keyboard scan, OS9 updates this bit pattern. A set bit (1) indicates that a key was pressed since the last scan. A clear bit (0) indicates that a key was not pressed. Definitions for the bits are as follows:

Bit	Key
0	<input type="checkbox"/> SHIFT
1	<input type="checkbox"/> CTRL or <input type="checkbox"/> CLEAR
2	<input type="checkbox"/> ALT or <input type="checkbox"/> @
3	<input type="checkbox"/> ↑ (up arrow)
4	<input type="checkbox"/> ↓ (down arrow)
5	<input type="checkbox"/> ← (left arrow)
6	<input type="checkbox"/> → (right arrow)
7	Space Bar

The bits can be masked with the following equates:

SHIFTBIT	equ	%00000001
CNTRLBIT	equ	%00000010
ALTERBIT	equ	%00000100
UPBIT	equ	%00001000
DOWNBIT	equ	%00010000
LEFTBIT	equ	%00100000
RIGHTBIT	equ	%01000000
SPACEBIT	equ	%10000000

- The support module for this call is CC3IO.

SS.ComSt

(Function code \$28). Return serial port configuration information

Entry Conditions:

A = *path number*
B = \$28

Exit Conditions:

Y = *high byte: parity*
 low byte: baud rate
(See the Setstat call SS.ComSt for values)

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- The SCF manager uses this call when performing an SS.Opt Getstat on an SCF-type device. User calls to SS.ComSt do not update the path descriptor. Use the SS.OPT Getstat call for most applications, because it automatically updates the path descriptor.

SS.MnSel

(Function code \$87). Requests that the high-level menu handler take control of menu selection

Entry Conditions:

A = *path number*
B = \$87

Exit Conditions:

A = *menu ID* (if valid selection)
0 (if invalid selection)
B = *item number of menu* (if valid selection)

Error Output:

CC = carry set on error
B = *error code* (if invalid selection)

Additional Information:

- After detecting a valid mouse click (when the mouse is pointing to a control area on a window), a process needs to call SS.MnSel. This call tells the enhanced window interface to handle any menu selection being made. If accumulator A returns with 0, then no selection has been made. The calling process needs to test and handle other returned values.
- A condition where Register A returns a valid menu ID number and Register B returns 0 signals the selection of a menu with no items. The application can now take over and do a special graphics pull down of its own. The menu title remains highlighted until the application calls the SS.UMBar SetStat to update the menu bar.
- The support module for this call is WindInt.

SS.Mouse

(Function code \$89). Gets mouse status

Entry Conditions:

- A = *path number*
- B = \$89
- X = *data storage area address*
- Y = *mouse port select:*
 - 0 = automatic selection
 - 1 = right side
 - 2 = left side

Exit Conditions:

- X = *data storage area address*

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- SS.Mouse returns information on the current mouse and its fire button. The following list defines the 32-byte data packet that SS.Mouse creates:

Pt.Valid	rmb 1	Is returned info valid? (0 = no, 1 = yes)
Pt.Actv	rmb 1	Active side (0 = off, 1 = right, 2 = left)
Pt.ToTm	rmb 1	Timeout initial value
Pt.TTTo	rmb 1	Time until timeout
	rmb 2	RESERVED
Pt.TSSt	rmb 2	Time since counter start
Pt.CBSA	rmb 1	Current button state (Button A)
Pt.CBSB	rmb 1	Current button state (Button B)
Pt.CCtA	rmb 1	Click count (Button A)
Pt.CCtB	rmb 1	Click count (Button B)
Pt.TTSA	rmb 1	Time this state counter (Button A)
Pt.TTSB	rmb 1	Time this state counter (Button B)
Pt.TLSA	rmb 1	Time last state counter (Button A)
Pt.TLSB	rmb 1	Time last state counter (Button B)
	rmb 2	RESERVED
Pt.BDX	rmb 2	Button down frozen Actual X
Pt.BDY	rmb 2	Button down frozen Actual Y
Pt.Stat	rmb 1	Window pointer type location
Pt.Res	rmb 1	Resolution (0-640 by 0 = 10/1 = 1)
Pt.AcX	rmb 2	Actual X value
Pt.AcY	rmb 2	Actual Y value
Pt.WRX	rmb 2	Window relative X
Pt.WRY	rmb 2	Window relative Y
Pt.Siz	equ .	Packet size 32 bytes
SPt.SRX	rmb 2	System use, screen relative X
SPt.SRY	rmb 2	System use, screen relative Y
SPt.Siz	equ .	Size of packet for system use

- Button Information:

Pt.Valid. The valid byte gives the caller an indication of whether the information contained in the returned packet is accurate. The information returned by this call is only valid if the process is running on the current interactive window. If the process is on a non-interactive window, the byte is zero and the process can ignore the information returned.

Pt.Actv. This byte shows which port is selected for use by all mouse functions. The default value is Right (1). You can change this value with the SS.GIP Setstat call.

Pt.ToTm. This is the initial value used by Pt.TTTo.

Pt.TTTo. This is the count down value (as of the instant the Getstat call is made). This value starts at the value contained in the Pt.ToTm and counts down to zero at a rate of 60 counts per second. The system maintains all counters until this value reaches 0, at which point it sets all counters and states to 0. The mouse scan routine changes into a quiet mode which requires less overhead than when the mouse is active. The timeout begins when both buttons are in the up (open) state. The timer is reinitialized to the value in Pt.ToTm when either button goes down (closed).

Pt.TSSst. This counter is constantly increasing, beginning when either button is pressed while the mouse is in the quiet state. All counts are a number of ticks (60 per second). The timer counts to \$FFFF, then stays at that value if additional ticks occur.

Pt.CBSA. These flag bytes indicate the state of the button **Pt.CBSB.** at the last system clock tick. A value of 0 indicates that the button is up (open). A value other than zero indicates that the button is down (closed). Button A is available on all Tandy joysticks and mice. Button B is only available for products that have two buttons.

The system scans the mouse buttons each time it scans the keyboard.

Pt.CCtA. This is the number of clicks that have occurred **Pt.CCtB.** since the mouse went into an active state. A click is defined as pressing (closing) the button, then releasing (opening) the button. The system counts the clicks as the button is released.

Pt.TTSA. This counter is the number of ticks that have **Pt.TTSB.** occurred during the current state, as defined by Pt.CBSx. This counter starts at one (counts the tick when the state changes) and increases by one for each tick that occurs while the button remains in the same state (open or closed).

Pt.TLSA. This counter is the number of ticks that have **Pt.TLSB.** occurred during the time that a button is in a state opposite of the current state. Using this count and the **T TSA/T TSB** count, you can determine how a button was in the previous state, even if the system returns the packet after the state has changed. Use these counters, along with the state and click count values, to define any type of click, drag, or hold convention you want.

Reserved. Two packet bytes are reserved for future expansion of the returned data.

- Position Information:

Pt.BDX. These values are copies of the **Pt.AcX** and **Pt.AcY** **Pt.BDY.** values when either of the buttons change from an open state to a closed state.

Pt.Stat. This byte contains information about the area of the screen on which the mouse is positioned. **Pt.Valid** must be a value other than 0 for this information to be accurate. If **Pt.Valid** is 0, this value is also 0 and not accurate. Three types of areas are currently defined:

- 0 = content region or current working area of the window
- 1 = control region (for use by Multi-View)
- 2 = off window, or on an area of the screen that is not part of the window

Pt.Res. This value is the current resolution for the mouse. The mouse must always return coordinates in the range of 0-639 for the X axis and 0-191 for the Y axis. If the system is so configured, you can use the high-resolution mouse adapter which provides a 1 to 1 ratio with these values plus 1. If the adapter is not in use, the resolution is a ratio of 1 to 10 on the X axis and 1 to 3 on the Y axis. The keyboard mouse provides a resolution of 1 to 1. The values in **Pt.Res** are:

- 0 = low res (x:10, y:3)
- 1 = high res (x,y:1)

Pt.AcX. The values read from the mouse returned in the **Pt.AcY.** resolution as described under **Pt.Res.**

Pt.WRX. The values read from the mouse minus the **Pt.WRY.** starting coordinates of the current window's working area. These values return the coordinates in numbers relative to the type of screen. For example, the X axis is in the range 0-639 for high-resolution screens and 0-319 for low resolution screens. You can divide the window relative values by 8 to obtain absolute character positions. These values are most helpful when working in non-scaled modes.

- The support modules for this call are CC3IO, GrfInt, and WindInt.

SS.Palet

(Function code \$91). Gets palette information

Entry Conditions:

- A = *path number*
- B = \$91
- X = *pointer to the 16-byte palette information buffer*

Exit Conditions:

- X = *pointer to the 16-byte palette information buffer*

Additional Information:

- SS.Palet reads the contents of the 16 screen palette registers, and stores them in a 16-byte buffer. When you make the call, be sure the X register points to the desired buffer location. The pointer is retained on exit. The palette values returned are specific to the screen on which the call is made.
- The support modules for this call are VDGINT, GrfInt, and WindInt.

SS.ScTyp

(Function code \$93). Returns the type of a screen to a calling program.

Entry Conditions:

A = *path*
B = \$93

Exit Conditions:

A = *screen type code*
1 = 40 x 24 text screen
2 = 80 x 24 text screen
3 = not used
4 = not used
5 = 640 x 192, 2-color graphics screen
6 = 320 x 192, 4-color graphics screen
7 = 640 x 192, 4-color graphics screen
8 = 320 x 192, 16-color graphics screen

Additional Information:

- Support modules for this system call are GrfInt and WindInt.

SS.FBRgs

(Function code \$96). Returns the foreground, background and border palette registers for a window.

Entry Conditions:

A = *path number*
B = \$96

Exit Conditions:

A = *foreground palette register number*
B = *background palette register number* (if carry clear)
X = *least significant byte of border palette register number*

Error Output:

B = *error code if any*
CC = *carry set on error*

Additional Information:

- Support modules for SS.FBRgs are GrfInt and WindInt.

SS.DFPal

(Function code \$97). Returns the default palette register settings.

Entry Conditions:

A = *path number*
B = \$97
X = *pointer to 16-byte data space*

Exit Conditions:

X = *default palette data moved to user space*

Error Output:

B = *error code, if any*
CC = *carry set on error*

Additional Information:

- You can use SS.DFPal to find the values of the default palette registers that are used when a new screen is allocated by GrfInt or WindInt. The corresponding SetStat can alter the default registers. This GetStat/SetStat pair is for system configuration utilities and should not be used by general applications.

Set Status System Calls

Use the Set Status system calls with the RBF manager subroutine SETSTA. The OS-9 Level Two system reserves function Codes 7-127 for use by Microware. You can define Codes 200-255 and their parameter-passing conventions for your own use. (See the sections on device drivers in Chapters 4, 5, and 6.)

Following are the Set Status functions and their codes.

SS.OPT

(Function code \$00). Writes the option section of the path descriptor

Entry Conditions:

- A = *path number*
- B = \$00
- X = *address of the status packet*

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- SS.OPT writes the option section of the path descriptor from the 32-byte status packet pointed to by Register X. Use this system call to set the device operating parameters, such as echo and line feed.

SS.SIZ

(Function code \$02). Changes the size of a file for RBF-type devices

Entry Conditions:

- A = *path number*
- B = \$02
- X = *most significant 16 bits of the desired file size*
- U = *least significant 16 bits of the desired file size*

Error Output:

- CC = carry set on error
- B = *error code* (if any)

SS.RESET

(Function code \$03). Restores the disk drive head to Track 0 in preparation for formatting and error recovery (use only with RBF-type devices)

Entry Conditions:

- A = *path number*
- B = \$03

Error Output:

- CC = carry set on error
- B = *error code* (if any)

SS.WTRK

(Function code \$04). Formats (writes) a track on a disk (RBF-type devices only)

Entry Conditions:

A = *path number*
B = \$04
U = *track number* (least significant 8 bits)
X = *address of the track buffer*
Y = *side/density*
 Bit B0 = *side*
 0 = Side 0
 1 = Side 1
 Bit B1 = *density*
 0 = single
 1 = double

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- For hard disks or floppy disks that have a “format entire diskette command,” SS.WTRK formats the entire disk only when the *track number* is zero.

SS.SQD

(**Function code \$0C**). Starts the shutdown procedure for a hard disk that has sequence-down requirements prior to removal of power. (Use only with RBF-type devices.)

Entry Conditions:

A = *path number*
B = \$0C

Exit Conditions: None

SS.KySns

(**Function code \$27**). Turns the key sense function on and off

Entry Conditions:

A = *path number*
B = \$27
X = *key sense switch value*
 0 = normal key operation
 1 = key sense operation

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- When SS.KySns switches the keyboard to key sense mode, the CC3IO module suspends transmission of keyboard characters to the SCF manager and the user. While the computer is in key sense mode, the only way to detect key press is with SS.KySns.
- The support module for this call is CC3IO.

SS.ComSt

(Function code \$28). Used by the SCF manager to configure a serial port

Entry Conditions:

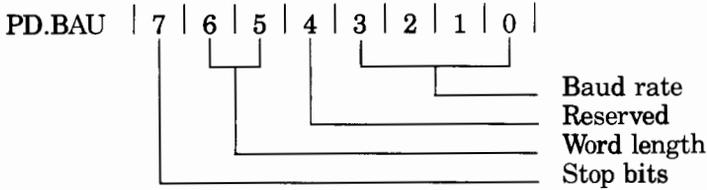
- A = path number
- B = \$28
- Y = high byte: parity
low byte: baud rate

Error Output:

- CC = carry set on error
- B = error code (if any)

Additional Information:

Baud Configuration. The high order byte of Y determines the baud rate, the word length, and number of stop bits. The byte is configured as follows:



Stop bits:

- 0 = 1
- 1 = 2

Word length:

- 00 = 8 bit
- 01 = 7 bit

Baud rate:

- 0000 = 110
- 0001 = 300
- 0010 = 600
- 0011 = 1200
- 0100 = 2400
- 0101 = 4800
- 0110 = 9600
- 0111 = 19200
- 1xxx = undefined

SS.AAGBf

(Function code \$80). Reserves an additional graphics buffer

Entry Conditions:

A = *path number*
B = \$80

Exit Conditions:

X = *buffer address*
Y = *buffer number (1-2)*

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- SS.AAGBf allocates an additional 8K graphics buffer. The first buffer (Buffer 0) must be allocated by using the DISPLAY GRAPHICS command. To use the DISPLAY GRAPHICS command, send control code \$0F to the standard terminal driver. SS.AAGBf can allocate up to two additional buffers (Buffers 1 and 2), one at a time.
- After calling SS.AAGBf, Register X contains the address of the new buffer. Register Y contains the buffer number.
- To deallocate all graphics buffers, use the END GRAPHICS control code.
- When SS.AAGBf allocates a buffer, it also maps the buffer into the application's address space. Each buffer uses 8K of the available memory in the application's address space. Also, if SS.DStat is called, Buffer 0 is also mapped into the application's address space. Allocation of all three buffers reduces the application's free memory by 24K.
- The support module for this call is VDGINT.

SS.SLGBf

(Function code \$81). Selects a graphics buffer

Entry Conditions:

- A = *path number*
- B = \$81
- X = \$00 *select buffer for use*
\$01-\$FF *select buffer for use and display*
- Y = *buffer number (0-2)*

Exit Conditions:

- X = *unchanged from entry*
- Y = *unchanged from entry*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- Use DISPLAY GRAPHICS to allocate the first graphics buffer. Use SS.AAGBf to allocate the second and third graphics buffers.
- Save each return address when writing directly to a screen. It is not necessary to save return addresses when using operating system graphics commands.
- SS.SLGBf does not update hardware information until the next vertical retrace (60Hz rate). Programs that use SS.AAGBf to change current draw buffers need to wait long enough to ensure that OS-9 has moved the current buffer to the screen.
- The screen shows the buffer only if the buffer is selected as the interactive device. If the device does not possess the keyboard, OS-9 stores the information until the device is selected as the interactive device. When the device is selected as the interactive device, the display shows the selected device's screen.
- The support module for this call is VDGINT.

SS.MpGPB

(Function code \$84). Maps the Get/Put buffer into a user address space

Entry Conditions:

- A = *path number*
- B = *\$84*
- X = *high byte: buffer group number*
low byte: buffer number
- Y = *action to take*
 - 1 = *map buffer*
 - 0 = *unmap buffer*

Exit Conditions:

- X = *address of the mapped buffer*
- Y = *number of bytes in buffer*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- The support modules for this call are GrfInt and WindInt.
- SS.MpGPB maps a Get/Put buffer into the user address space. You can then save the buffer to disk or directly modify the pixel data contained in the buffer. Use extreme care when modifying the buffer so that you do not write outside of the buffer data area.

SS.WnSet

(Function code \$86). Set up a high level window handler

Entry Conditions:

A = *path number*
B = \$86
X = *window data pointer* (if Y = WT.FSWin or WT.Win)
Y = *window type code*

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- The C language data structures for windowing are defined in the wind.h file in the DEFS directory of the system disk.
- The support module for this call is WindInt.

SS.SBar

(Function code \$88). Puts a scroll block at a specified position

Entry Conditions:

A = *path number*
B = \$88
X = *horizontal position of scroll block*
Y = *vertical position of scroll block*

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- WT.FSWin-type windows have areas at the bottom and right sides to indicate their relative positions within a larger area. These areas are called scroll bars. SS.SBar gives an application the ability to maintain relative position markers within the scroll bars. The markers indicate

the location of the current screen within a larger screen. Calling SS.SBar, updates both scroll markers.

- The support module for this call is WindInt.

SS.Mouse

(Function code \$89). Sets the sample rate and button timeout for a mouse

Entry Conditions:

A = *path number*
B = \$89
X = *mouse sample rate and timeout*
 most significant byte = *mouse sample rate*
 least significant byte = *mouse timeout*

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- SS. Mouse allows the application to define the mouse parameters. The sample rate is the number of clock ticks between the actual readings of the mouse status.
- The support module for the call is CC3IO.

SS.MsSig

(Function code \$8A). Sends a signal to a process when the mouse button is pressed

Entry Conditions:

- A = *path number*
- B = \$8A
- X = *user defined signal code* (low byte only)

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- SS.MsSig sends the process a signal the next time a mouse button changes state (from open to closed). Once SS.MsSig sends the signal, the process must repeat the Setstat each time that it needs to set up the signal.
- Processes using SS.MsSig should have an intercept routine to trap the signal. By intercepting the signal, other processes can be notified when the change occurs. Therefore, the other processes do not need to continually poll the mouse.
- The SS.Relea Setstat clears the pending signal request, if desired. It also clears any pending signal from SS.SSig. Because of this, if you want to clear only one signal, you must reset the other signal after calling SS.MsSig.
- The support module for this call is CC3IO.

SS.AScrn

(Function code \$8B). Allocates and maps a high-resolution screen into an application address space

Entry Conditions:

- A = *path number*
- B = *\$8B*
- X = *screen type*
 - 0 = 640 x 192 x 2 colors (16K)
 - 1 = 320 x 192 x 4 colors (16K)
 - 2 = 160 x 192 x 16 colors (16K)
 - 3 = 640 x 192 x 4 colors (32K)
 - 4 = 320 x 192 x 16 colors (32K)

Exit Conditions:

- X = *application address space of screen*
- Y = *screen number (1-3)*

Error Output:

- CC = *carry set on error*
- B = *error code (if any)*

Additional Information:

- SS.AScrn is particularly useful in systems with minimal memory when you want to allocate a high resolution graphics screen with all screen updating handled by a process.
- This call uses VDGInt (GRFINT is not required).
- All screens are allocated in multiples of 8K blocks. You can allocate a maximum of three buffers at one time. To select between buffers, use the SS.DScrn Setstat call.
- Screen memory is allocated but not cleared. The application using the screen must do this.
- Screens must be allocated from a VDG-type device—a standard 32-column text screen must be available for the device.
- The support module for this call is VDGINT.

SS.DScrn

(Function code \$8C). Causes VDGINT to display a screen that was allocated by SS.AScrn

Entry Conditions:

A = *path number*
B = \$8C
Y = *screen number* (1-3)

Error Output:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- SS.DScrn shows the requested screen if the requested screen is the current interactive device.
- The support module for this call is VDGINT.

SS.FScrn

(Function code \$8D). Frees the memory of a screen allocated by SS.AScrn

Entry Conditions:

A = *path number*
B = \$8D
Y = *screen number (1-3)*

Error Output:

CC = carry set on error
B = *error code (if any)*

Additional Information:

- Do not attempt to free a screen that is currently on the display.
- SS.FScrn returns the screen memory to the system and removes it from an application's address space.
- The support module for this call is VDGINT.

SS.PScrn

(Function code \$8E). Converts a screen to a different type

Entry Conditions:

- A = *path number*
- B = \$8E
- X = *new screen type*
 - 0 = 640 x 192 x 2 colors (16K)
 - 1 = 320 x 192 x 4 colors (16K)
 - 2 = 160 x 192 x 16 colors (16K)
 - 3 = 640 x 192 x 4 colors (32K)
 - 4 = 320 x 192 x 16 colors (32K)
- Y = *screen number*

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- SS.PScrn changes a screen allocated by SS.AScrn to a new screen type. You can change a 32K screen to either a 32K screen, or a 16K screen. You can change a 16K screen only to another 16K screen type. SS.PScrn updates the current display screen at the next clock interrupt.
- However, if you change a 32K screen to a 16K screen, OS-9 does not reclaim the extra 16K of memory. This means that you can later change the 16K screen back to a 32K screen.
- The support module for this call is VDGINT.

SS.Montr

(Function code \$92). Sets the monitor type

Entry Conditions:

- A = *path number*
- B = \$92
- X = *monitor type*
 - 0 = color composite
 - 1 = analog RGB
 - 2 = monochrome composite

Error Output:

- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- SS.Montr loads the hardware palette registers with the codes for the default color set for three types of monitors. The system default initializes the palette for a composite color monitor.
- The monochrome mode removes color information from the signals sent to a monitor.
- When a composite monitor is in use, a conversion table maps colors from RGB color numbers. In RGB and monochrome modes, the system uses the RGB color numbers directly.
- The support modules for this call are VDGINT and GrfDrv.

SS.GIP

(Function code \$94). Sets the system wide mouse and key repeat parameters

Entry Conditions:

- A = *path number*
B = \$94
X = *mouse resolution*; in the most significant byte
 0 = low resolution mouse
 1 = optional high resolution adapter
= *mouse port location*; in the least significant byte
 1 = right port
 2 = left port
Y = *key repeat start constant*; in the most significant byte
= *key repeat delay*; in the least significant byte
 00XX = no repeat
 FFFF = unchanged

Error Output:

- CC = carry set if error
B = *error code*, if any

Additional Information:

- Because this function affects system-wide settings, it is best to use it from system configuration utilities and not from general application program.
- The support module for this call is CC3IO.

SS.UMBAR

(Function code \$95). Requests the high level menu manager to update the menu bar.

Entry Conditions:

A = *path number*
B = \$95

Exit Conditions:

CC = carry set on error
B = *error code* (if any)

Additional Information:

- An application can call SS.UMBar when it needs to redraw menu bar information, such as when it enables or disables menus, or when it completes a window *pull down* and needs to restore the menu.
- The support module for this call is WindInt.

SS.DFPal

(Function code \$97). Sets the default palette register values

Entry Conditions:

- A = *path number*
- B = \$97
- X = *pointer to 16 bytes of palette data*

Exit Conditions:

- X = unchanged, bytes moved to system defaults
- CC = carry set on error
- B = *error code* (if any)

Additional Information:

- Use SS.DFPal to alter the system-wide palette register defaults. The system uses these defaults when it allocates a new screen using the DWSet command.
- Because this function affects system wide settings, it is best to use it from system configuration utilities, not general application programs.

SS.Tone

(Function code \$98). Creates a sound through the terminal output device.

Entry Conditions:

- A = *path number*
- B = \$98
- X = *duration and amplitude of the tone*
 - LSB = duration in ticks (60-sec) in the range 0-255
 - MSB = amplitude of tone in the range 0-63
- Y = *relative frequency counter* (0 = low, 4095 = high)

Exit Conditions:

These are the same as the entry conditions. There are no error conditions.

Additional Information:

- This call produces a programmed IO tone through the speaker of the monitor used by the terminal device. You can make the call on any valid path open to term or to a window device.
- The system does not mask interrupts during the time the tone is being produced.
- The frequency of the tone is a relative number ranging from 0 for a low frequency to 4095 for a high frequency. The widest variation of tones occurs at the high range of the scale.

Memory Module Diagrams

Executable Memory Module Format

Relative Address	Use	Check Range
\$00	Sync Bytes (\$87,\$CD)	
\$01		
\$02	Module Size (bytes)	
\$03		
\$04	Module Name Offset	header parity
\$05		
\$06	Type	Language
\$07	Attributes	Revision
\$08	Header Parity Check	module CRC
\$09	Execution Offset	
\$0A	Permanent Storage Size	
\$0B		
\$0C	(Additional optional header extensions located here) Module Body object code, constants, and so on	
\$0D		
	CRC Check Value	

Device Descriptor Format

Relative Address	Use	Check Range	
\$00	Sync Bytes (\$87,\$CD)	header parity	
\$01			
\$02	Module Size (bytes)		
\$03			
\$04	Offset to Module Name		
\$05			
\$06	\$F (Type)		\$1 (Lang)
\$07	Attributes		Revision
\$08	Header Parity Check		
\$09	Offset to File Manager Name String		Module CRC
\$0A			
\$0B	Offset to Device Driver Name String		
\$0D			
\$0E	Device Controller Absolute Physical Addr. (24 bit)		
\$0F			
\$10	Initialization Table Size		
\$11			
\$12,\$12 + n	(Initialization Table)		
	(Name Strings, and so on)		
	CRC Check Value		

INIT Module Format

Relative Address	Use	Check Range	
\$00	Sync Bytes (\$87,\$CD)	header parity	
\$01			
\$02	Module Size (bytes)		
\$03			
\$04	Module Name Offset		
\$05			
\$06	\$F (Type) \$1 (Lang)		
\$07	Attributes Revision		
\$08	Header Parity Check		Module CRC
\$09			
\$0A	Forced Limit of Top of Free RAM		
\$0B			
\$0C	#IRQ Polling Table Entries		
\$0D	#Device Table Entries		
\$0E			
\$0F	Offset to Startup Module Name String		
\$10			
\$11	Offset to Default Mass Storage Device Name String		
\$12			
\$13	Offset to Bootstrap Module Name String		
\$14-n	Name Strings		
	CRC Check Value		

Standard Floppy Disk Format

Color Computer 3

Physical Track Format Pattern

Format	Bytes (Dec)	Value (Hex)
Header pattern (once per track)	32	4E
	12	00
	3	F5
	1	FC
	32	4E
Sector pattern (repeated 18 times)	12	00
	3	F5
	1	track number (0-34)
	1	side number (0-1)
	1	sector number (1-18)
	1	sector length code (1)
	2	CRC
	22	4E
	12	00
	3	F5
	1	FB
	256	data area
	2	CRC
24	4E	
Trailer pattern (once per track)	N	4E (fill to index mark)

System Error Codes

The error codes are shown in both hexadecimal and decimal. The error codes listed include OS-9 system error codes, BASIC error codes, and standard windowing system error codes.

Code		Code Meaning
HEX	DEC	
\$01	001	UNCONDITIONAL ABORT — An error occurred from which OS-9 cannot recover. All processes are terminated.
\$02	002	KEYBOARD ABORT — You pressed BREAK to terminate the current operation.
\$03	003	KEYBOARD INTERRUPT — You pressed SHIFT BREAK either to cause the current operation to function as a background task with no video display or to cause the current task to terminate.
\$B7	183	ILLEGAL WINDOW TYPE — You tried to define a text type window for graphics or used illegal parameters.
\$B8	184	WINDOW ALREADY DEFINED — You tried to create a window that is already established.
\$B9	185	FONT NOT FOUND — You tried to use a window font that does not exist.
\$BA	186	STACK OVERFLOW — Your process (or processes) requires more stack space than is available on the system.
\$BB	187	ILLEGAL ARGUMENT — You have used an argument with a command that is inappropriate.
\$BD	189	ILLEGAL COORDINATES — You have given coordinates to a graphics command which are outside the screen boundaries.
\$BE	190	INTERNAL INTEGRITY CHECK — System modules or data are changed and no longer reliable.
\$BF	191	BUFFER SIZE IS TOO SMALL — The data you assigned to a buffer is larger than the buffer.

Code		Code Meaning
HEX	DEC	
\$C0	192	ILLEGAL COMMAND — You have issued a command in a form unacceptable to OS-9.
\$C1	193	SCREEN OR WINDOW TABLE IS FULL — You do not have enough room in the system window table to keep track of any more windows or screens.
\$C2	194	BAD/UNDEFINED BUFFER NUMBER — You have specified an illegal or undefined buffer number.
\$C3	195	ILLEGAL WINDOW DEFINITION — You have tried to give a window illegal parameters.
\$C4	196	WINDOW UNDEFINED — You have tried to access a window that you have not yet defined.
\$C8	200	PATH TABLE FULL — OS-9 cannot open the file, because the system path table is full.
\$C9	201	ILLEGAL PATH NUMBER — The path number is too large, or you specified a non-existent path.
\$CA	202	INTERRUPT POLLING TABLE FULL — Your system cannot handle an interrupt request, because the polling table does not have room for more entries.
\$CB	203	ILLEGAL MODE — The specified device cannot perform the indicated input or output function.
\$CC	204	DEVICE TABLE FULL — The device table does not have enough room for another device.
\$CD	205	ILLEGAL MODULE HEADER — OS-9 cannot load the specified module because its sync code, header parity, or Cyclic Redundancy Code is incorrect.
\$CE	206	MODULE DIRECTORY FULL — The module directory does not have enough room for another module entry.

Code		Code Meaning
HEX	DEC	
\$CF	207	MEMORY FULL — Process address space is full or your computer does not have sufficient memory to perform the specified task.
\$D0	208	ILLEGAL SERVICE REQUEST — The current program has issued a system call containing an illegal code number.
\$D1	209	MODULE BUSY — Another process is already using a non-shareable module.
\$D2	210	BOUNDARY ERROR — OS-9 has received a memory allocation or deallocation request that is not on a page boundary.
\$D3	211	END OF FILE — A read operation has encountered an end-of-file character and has terminated.
\$D4	212	RETURNING NON-ALLOCATED MEMORY — The current operation has attempted to deallocate memory not previously assigned.
\$D5	213	NON-EXISTING SEGMENT — The file structure of the specified device is damaged.
\$D6	214	NO PERMISSION — The attributes of the specified file or device do not permit the requested access.
\$D7	215	BAD PATH NAME — The specified pathlist contains a syntax error, for instance an illegal character.
\$D8	216	PATH NAME NOT FOUND — The system cannot find the specified pathlist.
\$D9	217	SEGMENT LIST FULL — The specified file is too fragmented for further expansion.
\$DA	218	FILE ALREADY EXISTS — The specified filename already exists in the specified directory.
\$DB	219	ILLEGAL BLOCK ADDRESS — The file structure of the specified device is damaged.

Code		Code Meaning
HEX	DEC	
\$DC	220	PHONE HANGUP-DATA CARRIER DETECT LOST — The data carrier detect is lost on the RS-232 port.
\$DD	221	MODULE NOT FOUND — The system received a request to link a module that is not in the specified directory.
\$DF	223	SUICIDE ATTEMPT — The current operation has attempted to return to the memory location of the stack.
\$E0	224	ILLEGAL PROCESS NUMBER — The specified process does not exist.
\$E2	226	NO CHILDREN — The system has issued a <i>wait service</i> request but the current process has no dependent process to execute.
\$E3	227	ILLEGAL SWI CODE — The system received a software interrupt code that is less than 1 or greater than 3.
\$E4	228	PROCESS ABORTED — The system received a signal Code 2 to terminate the current process.
\$E5	229	PROCESS TABLE FULL — A fork request cannot execute because the process table has no room for more entries.
\$E6	230	ILLEGAL PARAMETER AREA — A fork call has passed incorrect high and low bounds.
\$E7	231	KNOWN MODULE — The specified module is for internal use only.
\$E8	232	INCORRECT MODULE CRC — The CRC for the module being accessed is bad.
\$E9	233	SIGNAL ERROR — The receiving process has a previous, unprocessed signal pending.
\$EA	234	NON-EXISTENT MODULE — The system cannot locate the specified module.

Code		Code Meaning
HEX	DEC	
\$EB	235	BAD NAME — The specified device, file, or module name is illegal.
\$EC	236	BAD MODULE HEADER — The specified module header parity is incorrect.
\$ED	237	RAM FULL — No free system random access memory is available: the system address space is full, or there is no physical memory available when requested by the operating system in the system state.
\$EE	238	UNKNOWN PROCESS ID — The specified process ID number is incorrect.
\$EF	239	NO TASK NUMBER AVAILABLE — All available task numbers are in use.

Device Driver Errors

I/O device drivers generate the following error codes. In most cases, the codes are hardware-dependent. Consult your device manual for more details.

Code		Code Meaning
HEX	DEC	
\$F0	240	UNIT ERROR — The specified device unit doesn't exist.
\$F1	241	SECTOR ERROR — The specified sector number is out of range.
\$F2	242	WRITE PROTECT — The specified device is write-protected.
\$F3	243	CRC ERROR — A Cyclic Redundancy Code error occurred on a read or write verify.
\$F4	244	READ ERROR — A data transfer error occurred during a disk read operation, or there is a SCF (terminal) input buffer overrun.

Code		Code Meaning
HEX	DEC	
\$F5	245	WRITE ERROR — An error occurred during a write operation.
\$F6	246	NOT READY — The device specified has a <i>not ready</i> status.
\$F7	247	SEEK ERROR — The system attempted a seek operation on a non-existent sector.
\$F8	248	MEDIA FULL — The specified media has insufficient free space for the operation.
\$F9	249	WRONG TYPE — An attempt is made to read incompatible media (for instance an attempt to read double-side disk on single-side drive).
\$FA	250	DEVICE BUSY — A non-shareable device is in use.
\$FB	251	DISK ID CHANGE — You changed diskettes when one or more files are open.
\$FC	252	RECORD IS LOCKED-OUT — Another process is accessing the requested record.
\$FD	253	NON-SHAREABLE FILE BUSY — Another process is accessing the requested file.
\$FE	254	I/O DEADLOCK ERROR — Two processes have attempted to gain control of the same disk area at the same time.

Index

- ACIAPAK 8-135
- active process 2-12 - 2-13
 - queue 2-14, 8-98
 - state 2-13 - 2-14
- address
 - find 64K block 8-85
 - lines 2-7
 - polling 2-17
 - space, add module 8-104
- age, process 2-14
- alarm, set 8-66
- allocate
 - high RAM 8-69
 - image 8-70
 - memory 8-76
 - memory blocks 8-67 - 8-68
 - process descriptor 8-71
 - process task number 8-73
 - RAM 8-72
- allocation
 - bit map 8-7
 - map sector 5-1
 - of memory 2-5 - 2-7
 - polling 2-17
- allocation map
 - clear 8-13
 - disk 5-3
- alpha screen
 - cursor 8-118
 - memory 8-117
- ASM assembler 8-2
- assembler, RMA 8-2
- attach a device 8-44 - 8-45
- attribute
 - byte 5-5,
 - file 5-12
- background color, get 8-129
- bell, set alarm 8-66
- bit map 2-5
 - allocation 8-7
 - bit map (cont'd.)
 - search memory allocation 8-33
- block
 - allocate system memory 8-105
 - deallocate system memory 8-106
 - map into process space 8-96
 - number 2-7
 - scroll 8-139
- block map, system 8-18
- boot
 - file, load 5-26
 - module, link 8-75
- booting OS-9 1-3
- bootstrap
 - memory request 8-76
 - system 8-75
- border color, get 8-129
- buffer
 - map (Get/Put) 8-138
 - reserve graphics 8-136
- button
 - state, mouse 8-124 - 8-125, 8-126
 - timeout, mouse 8-140
- byte
 - attribute 5-5
 - deallocate 64-byte block 8-101
 - get from memory block 8-94
 - get two bytes 8-95
 - read from path 8-59 - 8-60
 - store in task 8-109
- calling process
 - insert in I/O queue 8-91
 - terminate 8-14
 - turn off 8-35, 8-43
- CC3DISK 1-2

- CC3GO module 2-19
- CC3IO 1-2, 6-1
- chain 8-8 - 8-9
- change
 - device operating parameters 5-23
 - directory 8-46
- character
 - read SCF input 6-13
 - write, SCF 6-14
- ChgDir 4-4
- child process 2-13
 - create 8-15 - 8-17
- clear specified block 8-77
- click 8-126
- CLOCK 1-2
- clock
 - module 1-2, 2-19
 - real-time 2-12, 2-17
- close
 - file 4-7
 - path 8-47, 8-135
- codes
 - signal 2-15
 - system error C-1
- command interpreter 1-4
- communication,
 - interprocess 2-15
- compact module directory 8-88
- compare strings 8-10
- compatibility with Level One 2-1
- concurrent execution 7-1 - 7-3
- copy external memory 8-11
- count, link 2-5
- counter start, mouse 8-124
- CPU 2-7
 - time 4-11
- CRC
 - calculate 8-12
 - validate module 8-111
 - value 3-1 - 3-3
- create
 - directory 8-55 - 8-56
 - create (cont'd.)
 - file 8-48 - 8-49
 - current
 - data directory 8-51
 - execution directory 8-51
 - cursor positioning 4-5
 - cyclic redundancy check 3-1 - 3-3
- DAT
 - hardware 8-99
 - registers 8-103
 - to logical address 8-78
- data
 - available, SCF test 8-113
 - directory 8-51
 - stream 4-3
 - transfer, pipes 7-1 - 7-3
 - move in memory 8-97
- DAT image 8-70
 - conversion 8-78
 - copy into process descriptor 8-102
 - deallocate block 8-77
 - high block 8-86
 - low block 8-87
 - pointer 8-95
- DAT task number
 - release 8-99
 - reserve 8-100
- date
 - get system 8-40
 - set 8-38
- deadlock 5-13
- deadly embrace 5-13
- deallocate
 - image RAM blocks 8-79
 - map bits 8-13
 - process descriptor 8-80
 - RAM blocks 8-81
 - task number 8-82
- default palette registers 8-129, 8-149
- delete file 8-50 - 8-51

-
- descriptions, system call 8-2
 - descriptor
 - get process 8-20
 - path 4-18
 - pointer 8-82
 - process 2-13
 - detach device 8-52
 - device
 - add or remove from
 - polling table 8-92
 - attach 8-44 - 8-45
 - attachment, verify 8-44 - 8-45
 - controller 5-15
 - control registers,
 - initialize 6-12
 - control registers, SCF 6-12
 - descriptor 1-4, 4-2, 4-17, A-2
 - detach 8-52
 - modules 5-15
 - modules, RBF 5-8 - 5-10
 - modules, SCF 6-6 - 6-8
 - name, get 8-115
 - open path to 8-57 - 8-58
 - operating parameters,
 - RBF 5-23
 - operating parameters, SCF 6-15
 - status 2-17 - 2-18, 8-63
 - status, get 8-54
 - table 4-2, 8-52
 - terminate, RBF 5-24
 - terminate, SCF 6-16
 - write to 8-64 - 8-65
 - device driver 1-3, 4-11
 - close path 8-135
 - modules 4-8
 - name 5-15
 - SCF 6-9 - 6-17
 - SCF subroutines 6-10 - 6-17
 - subroutines, RBF 5-16 - 5-27
 - device driver modules,
 - RBF 5-13 - 5-17
 - device interrupt 5-25
 - SCF 6-17
 - directory
 - attribute byte 5-5
 - change 8-46
 - disk 5-5
 - entry, module 8-83
 - get module 8-19
 - make 8-55 - 8-56
 - module 2-12, 8-88
 - disk
 - directories 5-5
 - sector read 5-19, 5-21
 - disk allocation map 5-3
 - sector 5-1
 - diskette format B-1
 - display
 - screen 8-143
 - status, get 8-115
 - drag 8-126
 - drive head, restore 8-131
 - duplicate path 8-53
 - editing, line 6-1, 8-61
 - end-of-file, test for 8-114
 - equate file 2-4
 - equivalent logical address 8-78
 - error
 - codes, system C-1 - C-6
 - message, write 8-30
 - print 8-30
 - exclamation point, pipes 7-1 - 7-3
 - execute
 - mode 5-11
 - system calls 8-1 - 8-2
 - execution
 - directory 8-51
 - offset, module 3-7
 - exit calling process 8-14
 - external memory, read 8-11
-

- fatal signal 2-13
- file
 - attribute byte 5-12
 - closing 4-7
 - create 4-4, 5-12, 8-48 - 8-49
 - deadlock 5-13
 - delete 4-5, 8-50 - 8-51
 - descriptor 5-3 - 5-4
 - execute mode 5-11
 - get pointer position 8-114
 - line reading/writing 4-6
 - load module 8-29
 - locking 5-12
 - non-shareable 5-12
 - opening 4-4
 - open path 8-57 - 8-58
 - permission bits 5-4
 - pipe 7-1 - 7-3
 - pointer 4-5, 8-62
 - position, RBF 8-114
 - read 5-1, 4-5
 - sharing 5-12
 - size, get 8-114
 - status, get 8-54, 8-114
 - update mode 5-11
 - write line to 8-64 - 8-65
 - writing 4-6
- file manager 1-3
 - modules 4-3
 - name 5-15
- find
 - 64-byte block 8-85
 - module directory
 - entry 8-84
- fire button 8-123 - 8-127
- FIRQ 4-12
 - interrupt 2-17
- flag, RAM In Use 8-81
- flip byte 2-17
- floppy diskette format B-1
- foreground color, get 8-129
- FORK 2-8
- fork, child process 8-15 - 8-17
- FORMAT 5-2
- format
 - device descriptor 4-17, A-2
 - INIT module A-3
 - memory module 3-6 - 3-7, A-1
 - of device driver
 - modules 4-10
 - track 8-132
- function
 - calls 2-4 - 2-5, 8-1
 - key sense 8-133
- get
 - a byte 8-94
 - free high block 8-86
 - free low block 8-87
 - ID 8-22
 - process pointer 8-89
 - status 8-54
 - Status system calls 8-112 - 8-130
 - system time 8-40
- Get/Put buffer, map 8-138
- GETSTA 8-112
 - SCF 6-15
- GetStat 4-6
- Getstats 5-23
- graphics buffer
 - reserve 8-136
 - select 8-137
- graphics interface 1-2
- GRFINT 1-2
- handler routine, virtual
 - interrupt 8-110
- hard disk shutdown 8-133
- hardware
 - controller, SCF 6-9
 - DAT registers 8-103
 - vector 2-16
- header
 - module 3-1 - 3-2
 - parity 8-111

-
- header (cont'd.)
 - pattern, floppy diskette B-1
 - high block, memory search 8-86
 - high-level
 - menu handler 8-122
 - menu manager 8-148
 - window handler 8-139
 - high-resolution
 - mouse adapter 8-126
 - screen, allocate 8-142
 - hold, button 8-126
 - I/O
 - calls 2-4 - 2-5, 8-1
 - device accessing 2-11
 - module, delete 8-90
 - path, close 8-47
 - queue, insert calling process 8-91
 - I/O system 1-3 - 1-4
 - calls 2-1, 8-2
 - system modules 1-1 - 1-4, 4-1
 - transfers 4-8
 - ID
 - return caller's process 8-22
 - set user 8-39
 - identification sector 5-1
 - image, allocate 8-70
 - INIT 1-2, 5-18
 - INIT module 2-17
 - format A-3
 - link 8-75
 - Init, SCF 6-12
 - initialization table, SCF
 - device 6-6 - 6-8
 - initialize device memory 5-18
 - input buffer, read SCF
 - character 6-13
 - insert process 8-74
 - install virtual interrupt 8-110
 - intercept, set signal 8-21
 - interface
 - graphics 1-2
 - VDG 4-2
 - Windint 4-2
 - interprocess
 - communication 2-15
 - interrupt
 - device 5-25
 - enable, SCF 6-12
 - FIRQ 2-17
 - processing 2-1
 - IOMAN 1-2
 - IRQ 4-12
 - add/remove device from polling table 8-92
 - interrupt 2-17
 - polling 2-17
 - polling table 2-18
 - service routine 5-25
 - IRQSVC routine 4-13
 - IRQSV 4-11
 - joystick value, get 8-116
 - kernel 1-2
 - key
 - repeat parameters, set 8-147
 - sense function 8-133
 - status, get 8-120
 - keyboard scan 2-17
 - language byte 3-4
 - line
 - editing 6-1, 8-61
 - reads 4-6, 8-61
 - writes 4-6, 8-65
 - link
 - to memory module 8-23 - 8-24, 8-28
 - using module directory entry 8-83
 - link count 2-5
 - decrease 8-42
-

- load
 - boot file 5-26
 - byte from memory
 - block 8-94
 - from task offset 8-93
 - module 8-25 - 8-26, 8-29
 - two bytes 8-95
- lock, end-of-lock 5-12
- locking
 - files 5-12
 - record 5-10 - 5-11
- logical
 - address space 2-6, 2-8
 - sector number 5-1
- LSN 5-2, 5-5
- macro 2-4
- MAKDIR 4-4
- make directory 8-55 - 8-56
- manager
 - file 1-3
 - random block 1-3
 - sequential file 1-3
- map
 - block 8-96
 - search allocation 8-33
- mask byte 2-18
- memory
 - allocate 8-76
 - allocate blocks 8-67 - 8-68
 - allocate high RAM 8-69
 - change process data
 - size 8-27
 - deallocate 2-5
 - find low block 8-87
 - free screen 8-144
 - map 2-6
 - module format 3-6 - 3-7, A-1
 - module, link 8-23 - 8-24
 - move data 8-97
 - page 2-5
 - pool 8-80
 - request, bootstrap 8-76
 - memory (cont'd.)
 - segment 2-8
 - memory allocation 2-5 - 2-7
 - memory block 2-7
 - find 64K 8-85
 - get byte 8-94
 - get high 8-86
 - map 8-81
 - map, search 8-72
 - memory management 2-1, 2-5 - 2-12
 - unit 2-7 - 2-8
 - menu
 - manager, update
 - request 8-148
 - selection 8-122
 - message, write error 8-30
 - MMU registers 2-8
 - mnemonic name, LSN 5-2
 - MODPAK 8-135
 - module
 - add into address
 - space 8-104
 - body 3-1 - 3-2
 - clock 2-19
 - CRC calculate 8-12
 - decrease link count 8-42
 - delete I/O module 8-90
 - device descriptor 5-15
 - device driver 4-8
 - file manager 4-3
 - finding 2-12
 - format 3-1 - 3-3
 - link 8-28
 - link count, decrease 8-42
 - linking 1-2
 - load 8-25 - 8-26, 8-29
 - load and execute
 - primary 8-8 - 8-9
 - name 3-3
 - RBF-type device
 - drivers 5-13 - 5-17
 - SCF device descriptor 6-6 - 6-8

- module (cont'd.)
 - types 3-1, 3-5
 - unlink 8-41
 - validate 8-111
- module directory 2-5, 2-12
 - compact 8-88
 - entry, link using 8-83
 - find 8-84
 - get 8-19
 - pointer 8-84
- module header 3-1 - 3-3, 5-15
 - SCF device driver 6-9
- monitor, set type 8-146
- mouse
 - button state 8-125
 - button timeout 8-140
 - click 8-122
 - coordinates 8-127
 - countdown 8-125
 - countup 8-125
 - parameters, set 8-147
 - port 8-125
 - resolution 8-126
 - screen position 8-126
 - send signal to process 8-141
 - status, get 8-123
 - timeout 8-124
 - window working area 8-127
- move data 8-97
- multiplexer 2-8
- multiprogramming 2-12 - 2-16
 - management 2-1
- multitasking 1-2
- name parse 8-31 - 8-32
- names, compare 8-10
- next process 8-98
- NMI interrupt 2-17
- non-shareable file 5-12
- number, path 8-53
- open
 - file 8-48 - 8-49
 - path 8-57 - 8-58
- operation of memory
 - management 2-8 - 2-12
- OS-9
 - Level One
 - compatibility 2-1
 - modules 1-2
 - scheduler 2-14 - 2-15
- OS9P3 2-1
 - module 2-2
- packet size 8-124
- palette, get information 8-127
- palette register 8-129
 - set default 8-149
 - settings 8-129
- parameters, mouse and key
 - repeat 8-147
- parent
 - directory 5-3
 - process 2-13
- parity 8-135
- parse name 8-31 - 8-32
- path
 - close 8-47, 8-135
 - duplicate 8-53
 - open 8-57 - 8-58
 - read bytes 8-59 - 8-60
 - table 4-2
- path descriptor 4-18, 5-5 - 5-8
 - read option section 8-112
 - SCF 6-2 - 6-6
 - write option section 8-130
- permanent storage size, module 3-7
- physical address space 2-7
- pipe file manager 4-3
- PIPEMAN 1-2 - 1-3, 4-3
- pipes 4-3, 7-1 - 7-3

- process descriptor 2-13 -
 - 2-14, 8-102
 - deallocate 8-80
 - descriptor, allocate 8-71
 - get 8-20
 - pointer 8-82
- processes
 - active 2-12
 - data size, change 8-27
- process ID 2-13
 - return caller's 8-22
- pseudo vector 2-16
- PutStat 4-6
- RAM 2-5 - 2-7
 - allocate 8-69, 8-72
 - allocate blocks 8-70
 - allocation 2-13
 - blocks, deallocate 8-81
 - blocks, deallocate
 - image 8-79
 - interrupt vector 2-18
- random
 - access 5-1
 - block file manager 1-3,
4-3
- RBF
 - change file size 8-131
 - format track 8-132
 - get file size 8-114
 - manager 4-3
 - tables 5-14 - 5-17
- read
 - bytes 8-59 - 8-60
 - device operating
 - parameters 5-23
 - disk sector 5-19
 - external memory 8-11
 - input character, SCF
 - 6-13
 - line 6-2, 8-61
 - mode 5-11
 - system call 6-1
- real-time clock 2-12, 2-17
- record locking 5-10
- reference
 - System Mode calls 8-5 -
8-6
 - User Mode system
calls 8-3 - 8-4
- registers
 - DAT 8-103
 - MMU 2-8
- release a task 8-99
- request system memory
8-105
- reserved memory 2-5 - 2-7
- reserve task number 8-100
- return
 - 64 bytes 8-101
 - system memory 8-106
- RMA assembler 8-2
- ROOT directory 5-3, 5-5
- RTS instruction 2-18
- SCF
 - configure serial port
8-134 - 8-135
 - data available test
8-113
 - device control
 - registers 6-12
 - Getsta 6-15
 - manager 4-3
 - path descriptor 6-2 - 6-6
 - terminate device 6-16
- scheduler, OS-9 2-14 - 2-15
- screen
 - allocate high-
resolution 8-142
 - convert type 8-145
 - display 8-143
 - free memory 8-144
 - mouse position 8-126
 - palette 8-127
 - size, get 8-119
 - type 8-128, 8-142, 8-145
- scroll block, install 8-139
- search bits 8-33

- sector 5-3
 - pattern, floppy diskette B-1
- seek, file pointer 8-62
- segment, memory 2-8
- select graphics buffer 8-137
- send signal 8-34
- sequential character
 - file manager 1-3, 4-3
 - I/O 6-1
- serial port configuration 8-121
- service
 - request processing 2-1
 - routine, IRQ 5-25
- set
 - alarm 8-66
 - date 8-38
 - IRQ 8-92
 - priority 8-36
 - process DAT image 8-102
 - process task DAT registers 8-103
 - status 8-63
 - SVC 8-107 - 8-108
 - SWI 8-37
 - time 8-38
 - user ID 8-39
- Setstats 5-23
- Set Status system calls 8-130 - 8-150
- shareable bit 3-5
- sharing, file 5-12
- shell 1-4
- shutdown hard disk 8-133
- signal 2-15 - 2-16
 - codes 2-15
 - fatal 2-13
 - from mouse to process 8-141
 - intercept trap 2-15 - 2-16
 - intercept, set 8-21
 - send to process 8-34
- single-user
 - attribute 5-12
 - bit, files 5-12
- size
 - of screen 8-119
 - of window 8-119
- sleep
 - calling process 8-35
- sleeping process 2-14, 2-16
- slices, time 2-12
- sound, create 8-150
- speaker, create sound 8-150
- state
 - active 2-13
 - of button 8-126
 - sleeping 2-14
 - suspend 4-13
 - waiting 2-13
- static storage address 2-18
- status
 - display 8-115
 - get, SCF 6-15
 - get mouse 8-123 - 8-127
 - of key 8-120
 - register 2-17
 - set, SCF 6-15
- status, get 8-54
- status, set 8-63
- store byte in a task 8-109
- string, scan input 8-31 - 8-32
- strings, compare 8-10
- subroutines
 - RBF device driver 5-16 - 5-27
 - SCF device drivers 6-10 - 6-17
- suspend
 - bit 4-13 - 4-14
 - state 4-13
- SWI, set 8-37
- SWI2 instruction 2-4
- symbolic names 2-4
- sync byte 3-3
- synonymous path number, return 8-53

- system
 - block map, get 8-18
 - boot 1-3
 - bootstrap 8-75
 - date, get 8-40
 - device, attach 8-44
 - error codes C-1 - C-6
 - initialization 2-1
 - link 8-104
 - mode call reference 8-5 - 8-6
 - time, get 8-40
- system call
 - add 8-107 - 8-108
 - descriptions 8-2, 2-4
 - execution 8-1 - 8-2
 - get status 8-112 - 8-130
 - mnemonics names 8-1
 - User Mode reference 8-3 - 8-4
- system memory
 - allocate high RAM 8-69
 - block map 8-81
 - deallocate 8-106
 - module directory, get 8-19
 - request 8-105
- system modules 1-1 - 1-4
- table
 - device 8-52
 - IRQ polling 2-18
 - RBF 5-14 - 5-17
 - SCF device descriptor 6-6 - 6-8
 - VIRQ 2-20
- task
 - map 2-12
 - offset, load from 8-93
 - register 2-8
 - release 8-99
 - store byte 8-109
- task number 8-73
 - DAT 8-100
 - deallocate 8-82
- terminal, create sound 8-150
- terminate
 - a device 5-24
 - calling process 8-14
 - SCF device 6-16
- ticks 4-11
- time
 - CPU 4-11
 - get system 8-40
 - set 8-38
 - sharing 2-11
 - slice 2-16, 2-12
- timeout, mouse 8-124
- track
 - format 8-132
 - restore drive head 8-131
- trailer pattern, floppy diskette B-1
- trap, signal intercept 2-15 - 2-16
- type
 - convert screen 8-145
 - of screen 8-128
 - set monitor 8-146
 - window screen 8-142
- unlink module 8-41 - 8-42
- update mode 5-11
- user calls 2-5
- user ID 2-13
 - set 8-39
- User Mode system calls reference 8-3 - 8-4
- validate module 8-111
- VDG 1-2
 - alpha screen cursor 8-118
 - alpha screen memory 8-117
 - interface 4-2
- vector
 - pseudo 2-16
 - set SWI 8-37
- vectoring 2-16

- verify device attachment
 - 8-44 - 8-45
- video display generator 1-2
- VIRQ 2-19 - 2-20
 - polling table 2-19 - 2-20
- virtual interrupt, install
 - 8-110
- wait
 - calling process 8-43
 - state 2-13 - 2-14
- waiting process 2-13
- wildcard 4-6
- WINDINT 1-2
- Windint interface 4-2
- window
 - descriptors 1-2
 - high-level handler 8-139
 - pointer location 8-124
 - screen, type 8-142
 - size, get 8-119
 - type 8-145
 - working area, mouse
 - 8-127
- working directory, change
 - 8-46
- write
 - character to SCF
 - output 6-14
 - disk sector 5-21
 - path descriptor 8-130 -
8-131
 - to file or device 8-64
- write line 8-65
 - line system call 6-2

Windows



OS-9
Windowing
System

Contents

Chapter 1 Types of OS-9 Windows	1-1
Device Windows	1-1
Overlay Windows	1-2
Opening a Device Window	1-2
Opening an Overlay Window	1-4
Chapter 2 Overview of Commands and Parameters	2-1
Parameters	2-1
Chapter 3 General Commands	3-1
Background Color	3-2
Bold Switch	3-3
Border Color	3-4
Change Working Area	3-5
Default Color	3-6
Define GET/PUT Buffer	3-7
Device Window End	3-10
Device Window Protect Switch	3-11
Device Window Set	3-12
Foreground Color	3-14
Select Font	3-15
Graphics Cursor Set	3-17
Get Block	3-18
GET/PUT Buffer Load	3-19
Kill GET/PUT Buffer	3-20
Logic Set	3-21
Overlay Window End	3-23
Overlay Window Set	3-24
Change Palette	3-26
Proportional Switch	3-28
Pattern Set	3-29
Put Block	3-32
Scale Switch	3-33
Window Select	3-34
Transparent Character Switch	3-35
Chapter 4 Drawing Commands	4-1
Draw Arc	4-2
Draw Bar	4-3
Relative Draw Bar	4-3

Draw Box	4-4
Relative Draw Box	4-4
Draw Circle	4-5
Draw Ellipse	4-6
Flood Fill	4-7
Draw Line	4-8
Relative Draw Line	4-8
Draw Line and Move	4-9
Relative Draw Line and Move	4-9
Draw Point	4-10
Relative Draw Point	4-10
Put Graphics Cursor	4-11
Set Draw Pointer	4-12
Relative Set Draw Pointer	4-12

Chapter 5 Text Commands	5-1
--------------------------------------	------------

Index

Types of OS-9 Windows

Unlike many operating systems, OS-9 has a built-in windows program. This driver, the Windowing System, lets you lay one or more smaller screen displays, called *windows*, on your screen display.

With these windows, you can perform several tasks at the same time. For example, suppose you are writing a business letter using a word processor in one window. You can go to a spreadsheet program in another window, get a price quote you need, return to the word processor, and include the price in the letter.

The Windowing System allows as many windows as your computer's memory can support, with a maximum of 32 at one time.

In OS-9, there are two types of windows: device and overlay.

Device Windows

A *device window* is one that can run a program or utility. This is the type of window you would use in the word processor/spreadsheet example given above. Each device window acts as an individual terminal.

The device windows are designated as devices `/w1 - /w7`. You open a device window as you do any other OS-9 device. You tell OS-9 the window's parameters—including whether the window is for text or graphics. If you want to run a process in the window, you can start an execution environment, such as a *shell*, on the window. (See "Opening a Device Window," later in this chapter, and the `DWSet` command in Chapter 3.)

Note: If you want only to send output to the device window—without running a process in the window—do not start a shell on the window.

Device windows cannot overlay each other, and their boundaries cannot overlap.

Overlay Windows

An *overlay window* is a window that you open on top of a device window. (You can place overlay windows over other overlay windows, but there must always be a device window at the bottom of the stack.) The purpose of overlay windows is to display computer dialog. You cannot fork a shell to an overlay window; however, you can **run** a shell in an overlay window. Overlay windows assume the screen type of the device windows they overlay.

Opening a Device Window

To open a device window, follow these steps:

1. If you want to allocate memory for the window, use OS-9's **iniz** command. Type:

```
iniz /wnumber 
```

where *number* is the number of the device window you wish to open (1-7). If you do not specify *number*, OS-9 uses the next available device window number.

If you do not use the **iniz** command, memory is allocated dynamically (as needed) to the window.

2. Next, you send an escape sequence to OS-9 that tells it the window's parameters. These parameters include the screen type, size, and colors. For example:

```
wcreate /w -s=2 20 10 40 10 01 00 00 
```

or

```
display 1b 20 02 14 0a 28 0a 01 00 00 /w  

```

sends the escape sequence for the next available window to the **DWSet** command. The **wcreate** command lets you use decimal numbers, while the **display** command requires hexadecimal numbers.

If you wish to send an escape sequence to a specific window, route the command to that device. For example:

```
wcreate /w2 -s=2 20 10 40 10 01 00 00 
```

sends the escape sequence to device `/w2`. The functions of the codes, as used in the `wcreate` command, are as follows:

- 2 sets a screen type of 80 x 24 (text only).
- 20 starts the window at character/column 20.
- 10 starts the window at line/row 10.
- 40 sets a window size of 40 characters.
- 10 sets a window size of 10 lines.
- 01 sets the foreground color to blue.
- 00 sets the background color to white.
- 00 sets the border to white.

If you do not send escape sequences, OS-9 uses default descriptors for the windows. The defaults are:

Window Number	Screen Type (chars./line)	Starting Position (horiz., vert.)	Size (columns, rows)
1	40 (text)	0,0	27,11
2	40 (text)	28,0	12,11
3	40 (text)	0,12	40,12
4	80 (text)	0,0	60,11
5	80 (text)	60,0	19,11
6	80 (text)	80,0	80,12
7	80 (text)	0,0	80,24

3. Use OS-9's `shell` command to fork a shell to the window. Type:

```
shell i=/wnumber & 
```

where `number` is the number used in the `iniz` or `wcreate` command. The `i=` parameter creates an *immortal* shell. Creating an immortal shell protects the window and its shell from being destroyed if you accidentally exit the shell using `CTRL BREAK`. If you omit the `i=` parameter, the shell is forked to the next available device window.

You now have a window that can run its own tasks. Information displayed in that window is automatically scaled to the window's size.

Opening an Overlay Window

To open an overlay window, use the Overlay Window Set function. (See OWSet in Chapter 3, "General Commands.")

Overview of Commands and Parameters

The windowing commands are divided among three chapters, based on their functions.

Chapter 3 describes the *general commands*. These commands let you create windows and buffers, access buffers, set switches, and maintain the window environment.

Chapter 4 describes the *drawing commands*. Besides letting you draw all kinds of images (circles, ellipses, arcs, and boxes, to name a few), these commands also enable you to color areas or to fill them with patterns.

Chapter 5 describes the *text commands*. Use these commands to manipulate the text cursor and the text attributes. Text commands operate on hardware text screens (Screen Types 1 and 2) and graphics windows if a font is selected.

Each command description lists the command's name, code, and parameters. To call a Windowing System command using OS-9's **display** command, type `display`, followed by the command code and the values you want to supply for the parameters.

Parameters

The following is a complete list of the parameter abbreviations used in Chapters 3, 4, and 5. All parameters represent a single byte of information.

Parameter	Description
HBX	<i>high order byte of x value</i>
LBX	<i>low order byte of x value</i>
HBY	<i>high order byte of y value</i>
LBX	<i>low order byte of y value</i>
HBXo	<i>high order byte of x-offset value (relative)</i>
LBXo	<i>low order byte of x-offset value (relative)</i>
HBYo	<i>high order byte of y-offset value (relative)</i>
LBYo	<i>low order byte of y-offset value (relative)</i>
HBR	<i>high order byte of radius</i>
LBR	<i>low order byte of radius</i>

Parameter	Description
HBL	<i>high order byte of length</i>
LBL	<i>low order byte of length</i>
HSX	<i>high order byte of size in x direction</i>
LSX	<i>low order byte of size in x direction</i>
HSY	<i>high order byte of size in y direction</i>
LSY	<i>low order byte of size in y direction</i>
HBRx	<i>high order byte of radius in x direction</i>
LBRx	<i>low order byte of radius in x direction</i>
GRP	<i>GET/PUT buffer group number (1-254)</i>
BFN	<i>GET/PUT buffer number (1-255)</i>
LCN	<i>logic code number</i>
PRN	<i>palette register number (0-15, wraps mod 15)</i>
CTN	<i>color table number (0-63, wraps mod 64)</i>
FNM	<i>font number</i>
CPX	<i>character position x (0-xmax)</i>
CPY	<i>character position y (0-ymax)</i>
STY	<i>screen type</i>
SVS	<i>save switch (0 = nosave, 1 = save area under overlay)</i>
SZX	<i>size in x (columns)</i>
SZY	<i>size in y (rows)</i>
XDR	<i>dimension ratio x used with YDR as YDR/XDR</i>
YDR	<i>dimension ratio y</i>
BSW	<i>binary switch (0 = off, 1 = on)</i>

General Commands

The general commands let you set up and customize windows. They also let you set up and access image buffers and select colors for the screen.

BColor Background Color

Function: Lets you choose a color palette register to use as the background color. See the Palette command for setting up the actual colors.

Code: 1B 33

Parameters: PRN

BoldSw Bold Switch

Function: Enables or disables boldfacing for text on graphics screens. If boldface is on, the screen displays subsequent characters in bold. If boldface is off, the screen displays subsequent characters in the regular font.

Code: 1B 3D

Parameters: BSW

BSW = switch
00 = off (Default)
01 = on

Notes:

- You can use BoldSw with any font.
- Boldface is not supported on hardware text screens (Screen Types 1 and 2).

Border Border Color

Function: Lets you change the palette register used for the screen border. See the Palette command for setting up the actual colors.

Code: 1B 34

Parameters: PRN

Notes:

- You set the border by selecting a palette register to use for the border register. When the actual color is changed in the palette register selected by the command, the color of the screen border changes to the new color. In general, the border register usually matches the background palette register.

CWArea Change Working Area

Function: Lets you alter the working area of the window. Normally, the system uses this call for high-level windowing, but you can use it to restrict output to a smaller area of the window.

Code: 1B 25

Parameters: CPX CPY SZX SZY

Notes:

- You cannot change a window's working area to be larger than the predefined area of the window as set by DWSet or OWSet.
- All drawing and window updating commands are done on the *current working area* of a window. The working area defaults to the entire size of the window. Scaling, when in use, is also performed relative to the current working area of a window. The CWArea command allows users to restrict the working area of a window to smaller than the full window size. Functions which might be done by opening a non-saved overlay window to draw or clear an image and then closing the overlay can be accomplished by using this command to shorten execution time where an actual overlay window is not needed.

DefColr Default Color

Function: Sets the palette registers back to their default values. The actual values of the palette registers depend on the type of monitor you are using. (See **montype** in *OS-9 Level Two Commands*.)

Code: 1B 30

Parameters: None

Notes:

- The default color definitions apply only to high-resolution graphics and text displays.
- The system sets the palette registers to a proper compatibility mode when switching to screens using the older VDG emulation modes. See the table below:

Window System Color Modes		VDG-Compatible Modes			
Palette	Color	P#	Color	P#	Color
00 & 08	White	00	Green	08	Black
01 & 09	Blue	01	Yellow	09	Green
02 & 10	Black	02	Blue	0A	Black
03 & 11	Green	03	Red	0B	Buff
04 & 12	Red	04	Buff	0C	Black
05 & 13	Yellow	05	Cyan	0D	Green
06 & 14	Magenta	06	Magenta	0E	Black
07 & 15	Cyan	07	Orange	0F	Orange

- The SetStat call lets you change the default color palette definition when using the windowing system. Default colors in the VDG-Compatible Mode cannot be changed. See the *OS-9 Level Two Technical Reference* manual for information on SetStat.
- The system's default colors are used whenever you create a new window.

DfnGPBuf Define GET/PUT Buffer

Function: Lets you define the size of the GET/PUT buffers for the system. Once you allocate a GET/PUT buffer, it remains allocated until you use the KilBuf command to delete it.

OS-9 allocates memory for GET/PUT buffers in 8K blocks that are then divided into the different GET/PUT buffers. Buffers are divided into buffer groups. Therefore, all commands dealing with GET/PUT buffers must specify both a group number and a buffer number within that group.

Code: 1B 29

Parameters: GRP BFN HBL LBL

Technical:

The buffer usage map is as follows:

Group Number	Buffer Number ¹	Use
0	1 - 255	Internal use only (returns errors)
1 - 199	1 - 255	General use by applications ²
200 - 254	1 - 255	Reserved (Microware use only) ³
255	1 - 255	Internal use only (returns errors)

¹ Buffer Number 0 is invalid and cannot be used.

² The application program should request its user ID via the GetID system call to use as its group number for buffer allocation.

³ The standard group numbers are defined as follows:

Note: The names, buffer groups, and buffer numbers are defined in the assembly definition file. The decimal number you use to call these are in parentheses next to the name. For example, to select the Arrow pointer, Grp_Ptr and Ptr_Arr, you use 202,1 as the group/buffer number.

- Grp_Fnt(200) = font group for system fonts
 - Fnt_S8x8(1) = standard 8x8 font
 - Fnt_S6x8(2) = standard 6x8 font
 - Fnt_G8x8(3) = standard graphics font

The standard fonts are in the file SYS/StdFonts.

- Grp_Clip(201) = clipboarding group (for Multiview)

- Grp_Ptr(202) = graphics cursor (pointer) group
 - Ptr_Arr(1) = arrow pointer (hp=0,0)
 - Ptr_Pen(2) = pencil pointer (hp=0,0)
 - Ptr_LCH(3) = large cross hair pointer (hp=7,7)
 - Ptr_Slp(4) = sleep indicator (hourglass)
 - Ptr_Ill(5) = illegal indicator
 - Ptr_Txt(6) = text pointer (hp=3,3)
 - Ptr_SCH(7) = small cross hair pointer (hp=3,3)

hp = hit point, the coordinates of the actual point on the object at which the cursor should be centered.

The standard pointers are in the file SYS/StdPtrs.

- Grp_Pat2(203) = two color patterns
- Grp_Pat4(204) = four color patterns
- Grp_Pat6(205) = sixteen color patterns

- Pat_Dot(1) = dot pattern
- Pat_Vrt(2) = vertical line pattern
- Pat_Hrz(3) = horizontal line pattern
- Pat_XHtc(4) = cross hatch pattern
- Pat_LSnt(5) = left slanted lines
- Pat_RSnt(6) = right slanted lines
- Pat_SDot(7) = small dot pattern
- Pat_BDot(8) = large dot pattern

Each pattern is found within each of the pattern groups.

Standard patterns are in the files SYS/StdPats_2, SYS/StdPats_4, and SYS/StdPats_16.

All files have GPLoad commands imbedded in file, along with the data. To load fonts, pointers, or patterns, simply merge them to any window device: For example:

```
merge SYS/StdFonts 
```

sends the standard font to standard output which may be redirected to another device if the current output device is not a window device (such as when term is a VDG screen).

You only need to load fonts once for the entire system. Once a Get/Put buffer is loaded, it is available to all devices and processes in the system.

DWEnd Device Window End

Function: Ends a current device window. DWEnd closes the display window. If the window was the last device window on the screen, DWEnd also deallocates the memory used by the window. If the window is an interactive window, OS-9 automatically switches you to a new device window, if one is available.

Code: 1B 24

Parameters: None

Notes:

- DWEnd is only needed for windows that have been attached via the `iniz` utility or the `I$Attach` system call. Non-attached windows have an implied DWEnd command that is executed when you close the path.

DWProtSw Device Window Protect Switch

Function: Disables and enables device window protection. By default, device windows are protected so that you cannot overlay them with other device windows. This type of protection helps avoid the possibility of destroying the contents of either or both windows.

Code: 1B 36

Parameters: BSW

BSW = switch
00 = off
01 = on (Default)

Notes:

- We recommend that you not turn off device window protection. If you do, however, use extreme discretion because you might destroy the contents of the windows. OS-9 does not return an error if you request that a new window be placed over an area of the screen which is already in use by an unprotected window.

DWSet Device Window Set

Function: Lets you define a window's size and location on the physical screen. Use DWSet after opening a path to a device window.

Code: 1B 20

Parameters: STY CPX CPY SZX SZY PRN1 PRN2 PRN3

PRN1 = Foreground
PRN2 = Background
PRN3 = Border (if STY \geq 1)

Notes:

- The **iniz** and **display** commands open paths to the device window.
- When using DWSet in a program, you must first open the device.
- Output to a new window is ignored until OS-9 receives a DWSet command, unless defaults are present in the device descriptor (/w1-/w7). If defaults are present in the device descriptors, OS-9 automatically executes DWSet, using those defaults.
- When OS-9 receives the DWSet, it allocates memory for the window, and clears the window to the current background color. If the standard font is already in memory, OS-9 assigns it as the default font. If the standard font is not in memory, you must execute a font set (Font) command after loading the fonts to produce text output on a graphics window.
- Use the Screen Type code (STY) to define the resolution and color mode of the new screen. If the screen type code is zero, OS-9 opens the window on the process's currently selected screen. If the code is 01, OS-9 opens the window on the currently displayed screen. If the code is non-zero, OS-9 allocates a new screen for the window. The following describes the acceptable screen types:

Code	Screen Size	Colors	Memory	Type
FF	Current Displayed Screen ¹			
00	Process's Current Screen			
01	40 x 24	8 & 8	2000	Text
02	80 x 24	8 & 8	4000	Text
05	640 x 192	2	16000	Graphics
06	320 x 192	4	16000	Graphics
07	640 x 192	4	32000	Graphics
08	320 x 192	16	32000	Graphics

¹ Use the Current Displayed Screen option only in procedure files to display several windows on the same physical screen. All programs should operate on that process's current screen.

- The location of the window on the physical screen is determined by the diagonal line defined by:

(CPX,CPY) and (CPX + SZX, CPY + SZY)

- The foreground, background, and border register numbers (PRN1, PRN2, and PRN3) define the palette registers used for the foreground and background colors. See the Palette command in this chapter for more information.
- When an implicit or explicit DWSet command is done on a window, the window automatically clears to the background color.
- All windows on the screen must be of the same type (either text or graphics).
- Values in the palette register affect all windows on the screen. However, you can choose which register to use for foreground and background for each window. That is, OS-9 maintains palette registers and border register numbers for the entire screen and foreground and background register numbers for each individual window.
- OS-9 deallocates memory for a screen when you terminate the last window on that screen.

FColor Foreground Color

Function: Lets you select a color palette register for the foreground color. See the Palette command for setting the actual colors.

Code: 1B 32

Parameters: PRN

Font Select Font

Function: Lets you select/change the current font. Before you can use this command, the font must be loaded into the specified GET/PUT group and buffer (using GPLoad). See the GPLoad command for information on loading font buffers.

Code: 1B 3A

Parameters: GRP BFN

Notes:

- You can select proportional spacing for the font by using PropSw.
- All font data is a 2-color bit map of the font.
- Each character in the font data consists of 8 bytes of data. The first byte defines the top scan line, the second byte defines the second scan line, and so on. The high-order bit of each byte defines the first pixel of the scan line, the next bit defines the next pixel, and so on. For example, the letter "A" would be represented like this:

Byte	Pixel Representation
10	. . . #
28	. . # . # . . .
44	. # . . . # . .
44	. # . . . # . .
7c	. # # # # # . .
44	. # . . . # . .
44	. # . . . # . .
00

Note that 6x8 fonts ignore the last 2 bits per byte.

- The fonts are ordered with characters in the following ranges:

\$00-\$1F	International characters (see mapping below)
\$20-\$7F	Standard ASCII characters

International characters or any characters in the font below character \$20 (hex) are printed according to the following table:

Character position in font	Char1	or	Char2
\$00	\$C1		\$E1
\$01	\$C2		\$E2
\$02	\$C3		\$E3
\$03	\$C4		\$E4
\$04	\$C5		\$E5
\$05	\$C6		\$E6
\$06	\$C7		\$E7
\$07	\$C8		\$E8
\$08	\$C9		\$E9
\$09	\$CA		\$EA
\$0A	\$CB		\$EB
\$0B	\$CC		\$EC
\$0C	\$CD		\$ED
\$0D	\$CE		\$EE
\$0E	\$CF		\$EF
\$0F	\$D0		\$F0
\$10	\$D1		\$F1
\$11	\$D2		\$F2
\$12	\$D3		\$F3
\$13	\$D4		\$F4
\$14	\$D5		\$F5
\$15	\$D6		\$F6
\$16	\$D7		\$F7
\$17	\$D8		\$F8
\$18	\$D9		\$F9
\$19	\$DA		\$FA
\$1A	\$AA		\$BA
\$1B	\$AB		\$BB
\$1C	\$AC		\$BC
\$1D	\$AD		\$BD
\$1E	\$AE		\$BE
\$1F	\$AF		\$BF

GCSet Graphics Cursor Set

Function: Creates a GET/PUT buffer for defining the graphics cursor that the system displays. You must use GCSet to display a graphic cursor.

Code: 1B 39

Parameters: GRP BFN

Notes:

- To turn off the graphics cursor specify GRP as 00.
- A system standard buffer or a user-defined buffer can be used for the graphics cursor.

GetBlk Get Block

Function: Saves an area of the screen to a GET/PUT buffer. Once the block is saved, you can put it back in its original location or in another on the screen, using the PutBlk command.

Code: 1B 2C

Parameters: GRP BFN HBX LBX HBY LBY HSX LSX HSY LSY

HBX/LBX = *x-location of block* (upper left corner)

HBY/LBY = *y-location of block*

HSX/LSX = *x-dimension of block*

HSY/LSY = *y-dimension of block*

Notes:

- The GET/PUT buffer maintains information on the size of the block stored in the buffer so that the PutBlk command works more automatically.
- If the GET/PUT buffer is not already defined, GetBlk creates it. If the buffer is defined, the data must be equal to or smaller than the original size of the buffer.

GPLoad GET/PUT Buffer Load

Function: Preloads GET/PUT buffers with images that you can move to the screen later, using PutBlk.

If the GET/PUT buffer is not already created, GPLoad creates it.

If the buffer was previously created, the size of the passed data must be equal to or smaller than the original size of the buffer. Otherwise, GPLoad truncates the data to the size of the buffer.

Code: 1B 2B

Parameters: GRP BFN STY HSX LSX HSY LSY HBL LBL
(Data...)

STY = *format*

HSX/LSX = *x-dimension of stored block*

HSY/LSY = *y-dimension of stored block*

HBL/LBL = *number of bytes in data*

Notes:

- Buffers are maintained in a linked list system.
- Buffers to be used most should be allocated last to minimize the search time in finding the buffers.
- When loading a Font GET/PUT Buffer, the parameters are as follows:

GRP BFN STY HSX LSX HSY LSY HBL LBL
(Data...)

GRP = 254

STY = 5

HSX/LSX = *x-dimension size of Font 6 or 8*

HSY/LSY = *y-dimension size of Font 8*

HBL/LBL = *size of font data (not including this header information)*

See the Font command for more information on font data.

KilBuf Kill GET/PUT Buffer

Function: Deallocates the buffer specified by the group and buffer number. To deallocate the entire group of buffers, set the buffer number to 0.

Code: 1B 2A

Parameters: GRP BFN

Notes:

- KilBuf returns memory used by the buffer to a free list. When an entire block of memory has been put on the free list, the block is returned to the system.

LSet Logic Set

Function: Lets you create special effects by specifying the type of logic used when storing data, which represents an image, to memory. The specified logic code is used by all draw commands until you either choose a new logic or turn off the logic operation. To turn off the logic function, set the logic code to 00.

Code: 1B 2F

Parameters: LCN

LCN = *logic code number*

00 = No logic code; store new data on screen

01 = AND new data with data on screen

02 = OR new data with data on screen

03 = XOR new data with data on screen

Notes:

- The following tables summarize logic operations in bit manipulations:

AND	First Operand	Second Operand	Result
	1	1	1
	1	0	0
	0	1	0
	0	0	0

OR	First Operand	Second Operand	Result
	1	1	1
	1	0	1
	0	1	1
	0	0	0

XOR	First Operand	Second Operand	Result
	1	1	0
	1	0	1
	0	1	1
	0	0	0

- Data items are represented as palette register numbers in memory. Since logic is performed on the palette register number and not the colors in the registers, you should choose colors for palette registers carefully so that you obtain the desired results. You may want to choose the colors for the palette registers so that LSet appears to *and*, *or*, and *xor* the colors rather than the register numbers. For example:

Palette #	Color	Alternative Order
0	White	Black
1	Blue	Blue
2	Black	Green
3	Green	White

OWEnd Overlay Window End

Function: Ends a current overlay window. OWEnd closes the overlay window and deallocates memory used by the window. If you opened the window with a *save switch* value of hexadecimal 01, OS-9 restores the area under the window. If you did not, OS-9 does not restore the area and any further output is sent to the next lower overlay window or to the device window, if no overlay window exists.

Code: 1B 23

Parameters: None

OWSet Overlay Window Set

Function: Use OWSet to create an overlay window on an existing device window. OS-9 reconfigures current device window paths to use a new area of the screen as the current logical device window.

Code: 1B 22

Parameters: SVS CPX CPY SZX SZY PRN1 PRN2

SVS = *save switch*

00 = Do not save area overlaid

01 = Save area overlaid and restore at close

PRN1 = *background palette register*

PRN2 = *foreground palette register*

Notes:

- If you set SVS to zero, any writes to the new overlay window destroy the area under the window. You might want to set SVS to zero if your system is already using most of its available memory. You might also set SVS to zero whenever it takes relatively little time to redraw the area under the overlay window once it is closed.
- If you have ample memory, specify SVS as 1. Doing this causes the system to save the area under the new overlay window. The system restores the area when you terminate the new overlay window. (See OWEnd.)
- The size of the overlay window is specified in standard characters. Use the same resolution (number of characters) as the device window that will reside beneath the overlay window. Have your program determine the original size of the device window at startup (using the SS.ScSiz GETSTAT call), if the device window does not cover the entire screen. See the *OS-9 Level Two Technical Reference* manual for information on the SS.ScSiz GETSTAT call.

- Overlay windows can be created on top of other overlay windows; however, you can only write to the top most window. Overlay windows are “stacked” on top of each other logically. To get back down to a given overlay, you must close (OWEnd) any overlay windows that reside on top of the desired overlay window.
- Stacked overlay windows do not need to reside directly on top of underlying overlay windows. However, all overlay windows must reside within the boundaries of the underlying device window.

Palette Change Palette

Function: Lets you change the color associated with each of the 16 palette registers.

Code: 1B 31

Parameters: PRN CTN

Notes:

- Changing a palette register value causes all areas of the screen using that palette register to change to the new color. In addition, if the border is set to that palette register, the border color also changes. See the Border command for more information.
- Colors are made up by setting the red, green, and blue bits in the color byte which is inserted in the palette register. The bits are laid out as follows:

Bit	Color
0	Blue low
1	Green low
2	Red low
3	Blue high
4	Green high
5	Red high
6	unused
7	unused

By using six bits for color (2 each for red, green and blue) there is a possibility of 64 from which to choose. Some of the colors are defined as shown:

White	: 00111111 = \$3F (all color bits set)
Black	: 00000000 = \$00 (no color bits set)
Standard Blue	: 00001001 = \$09 (both blue bits set)
Standard Green	: 00010010 = \$12 (both green bits set)
Standard Red	: 00100100 = \$24 (both red bits set)

Note: These colors are for RGB monitors. The composite monitors use a different color coding and do not directly match pure RGB colors. To get composite color from the RGB colors, the system uses conversion tables. The colors were assigned to match the RGB colors as close as possible. There are, however, a wider range of composite colors, so the colors without direct matches were assigned to the closest possible match. The white, black, standard green, and standard orange are the same in both RGB and composite.

PropSw Proportional Switch

Function: Enables and disables the automatic proportional spacing of characters. Normally, characters are not proportionally spaced.

Code: 1B 3F

Parameters: BSW

BSW = *switch*
00 = off (Default)
01 = on

Notes:

- Any standard software font used in a graphics screen can be proportionally spaced.
- Proportional spacing is not supported on hardware text screens.

PSet Pattern Set

Function: Selects a preloaded GET/PUT buffer as a pattern RAM array. This pattern is used with all draw commands until you either change the pattern or turn it off by passing a parameter of 00 as GRP (Group Number).

Code: 1B 2E

Parameters: GRP BFN

Notes:

- The pattern array is a 32 x 8 pixel representation of graphics memory. The color mode defines the number of bits per pixel and pixels per byte. So, be sure to take the current color mode into consideration when creating a pattern array.
- The GET/PUT buffer can be of any size, but only the number of bytes as described by the following table are used:

Color Mode	Size of Pattern Array
2	4 bytes x 8 = 32 bytes (1 bit per pixel)
4	8 bytes x 8 = 64 bytes (2 bits per pixel)
16	16 bytes x 8 = 128 bytes (4 bits per pixel)

- The buffer must contain at least the number of bytes required by the current color mode. If the buffer is larger than required, the extra bytes are ignored.
- To turn off patterning, set GRP to 00.

PutBlk Put Block

Function: Moves a GET/PUT buffer, previously copied from the screen or loaded with GPLoad, to an area of the screen.

Code: 1B 2D

Parameters: GRP BFN HBX LBX HBY LBY

HBX/LBX = *x-location of block*
(upper left corner)

HBY/LBY = *y-location of block*

Notes:

- The dimensions of the block were saved in the GET/PUT buffer when you created it. OS-9 uses these dimensions when restoring the buffer.
- The screen type conversion is automatically handled by the PutBlk routine in the driver.
- GET/PUT buffers cannot be scaled. The image will be clipped if it does not fit within the window.

ScaleSw Scale Switch

Function: Disables and enables automatic scaling. Normally, automatic scaling is enabled. When scaling is enabled, coordinates refer to a relative location in a window that is proportionate to the size of the window. When scaling is disabled, coordinates passed to a command will be the actual coordinates for that type of screen relative to the origin of the window.

Code: 1B 35

Parameters: BSW

BSW = *switch*
00 = off
01 = on (Default)

Notes:

- A useful application of disabled scaling is the arrangement of references between a figure and text.
- All coordinates are relative to the window's origin (0,0). The valid range for the coordinates:

Scaling enabled:

y = 0-191
x = 0-639

Scaling disabled:

y = 0-*size of y* - 1
x = 0-*size of x* - 1

Select Window Select

Function: Causes the current process's window to become the active (display) window. You can select a different window by using the form:

```
display 1B 21 >/wnumber
```

where *number* is the desired window number. If the process that executes the select is running on the current interactive (input/display) window, the selected window becomes the interactive window, and the other window becomes passive.

Code: 1B 21

Parameters: None

Notes:

- The keyboard is attached to the process's selected window through the use of the **CLEAR** key. This lets you input data from the keyboard to different windows by using the **CLEAR** key to select the window.
- All display windows that occupy the same screen are also displayed.
- The device window which owns the keyboard is the current interactive window. The interactive window is always the window being displayed. Only one process may receive input from the keyboard at a time. Many processes may be changing the output information on their own windows; however, you can only see the information that is displayed on the interactive window and any other window on the same screen as the interactive window.

TCharSw Transparent Character Switch

Function: Defines the character mode to be used when putting characters on the graphics screens.

In the default mode (transparent off), the system uses block characters that draw the entire foreground and background for that cell.

When in transparent mode, the only pixels that are changed are the ones where the character actually has pixels set in its font. When transparent mode is off, all pixels in the character block are set: foreground or font pixels in the foreground color and others in the background color.

Code: 1B 3C

Parameters: BSW

BSW = *switch*
00 = off (Default)
01 = on



Drawing Commands

All drawing commands relate to an invisible point of reference on the screen called the *draw pointer*. Originally, the draw pointer is at position 0,0. You can change the position by using the SetDPtr and RSetDPtr commands described in this chapter. In addition, some draw commands automatically update the draw pointer. For example, the LineM command draws a line from the current draw pointer position to the specified end coordinates and moves the draw pointer to those end coordinates. The Line command draws a line but does not move the pointer.

Also, note that all draw commands are affected by the pattern and logic commands described in Chapter 3.

Do not confuse the draw pointer with the graphics cursor. The graphics cursor is the graphic representation of the mouse/joystick position on the screen.

In this chapter, commands that use relative coordinates (*offsets*) are listed with their counterparts that use absolute coordinates. For example, RSetDPtr is listed under SetDPtr.

Arc3P Draw Arc

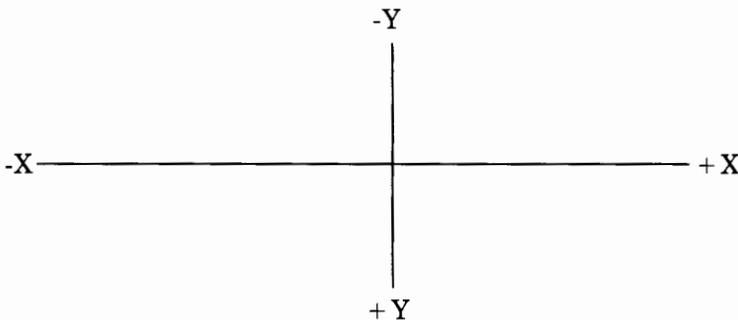
Function: Draws an arc with its midpoint at the current draw pointer position. You specify the curve by both the X and Y dimensions, as you do an ellipse. In this way, you can draw either elliptical or circular arcs. The arc is clipped by a line defined by the (X01,Y01) - (X02,Y02) coordinates. These coordinates are signed 16 bit values and are relative to the center of the ellipse. The draw pointer remains in its original position.

Code: 1B 52

Parameters: HBRx LBRx HBRy LBRy HX01 LX01 HY01 LY01 HX02 LX02 HY02 LY02

Notes:

- The resulting arc depends on the order in which you specify the line coordinates. Arc3P first draws the line from Point 1 to Point 2 and then draws the ellipse in a clockwise direction.
- The coordinates of the screen are as follows:



Bar Draw Bar

Function: Draws and fills a rectangle that is defined by the diagonal line from the current draw pointer position to the specified position. The box is drawn in the current foreground color. The draw pointer returns to its original location.

Code: 1B 4A

Parameters: HBX LBX HBY LBY

RBar Relative Draw Bar

Function: Draws and fills a rectangle that is defined by the diagonal line from the current draw pointer position to the point specified by the offsets. The box is drawn in the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 4B

Parameters: HBX_o LBX_o HBY_o LBY_o

BOX Draw Box

Function: Draws a rectangle that is defined by the diagonal line from the current draw pointer position to the specified position. The box is drawn in the current foreground color. The draw pointer returns to its original location.

Code: 1B 48

Parameters: HBX LBX HBY LBY

RBOX Relative Draw Box

Function: Draws a rectangle that is defined by the diagonal line from the current draw pointer position to the point specified by the offsets. The box is drawn in the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 49

Parameters: HBX₀ LBX₀ HBY₀ LBY₀

Circle Draw Circle

Function: Draws a circle of the specified radius with the center of the circle at the current draw pointer position. The circle is drawn in the current foreground color. The draw pointer remains in its original location.

Code: 1B 50

Parameters: HBR LBR

Ellipse Draw Ellipse

Function: Draws an ellipse with its center at the current draw pointer position. The X value specifies the horizontal radius, and the Y value specifies the vertical radius. The ellipse is drawn in the current foreground color. The draw pointer remains in its original location. This is a relative command.

Code: 1B 51

Parameters: HBRx LBRx HBRy LBRy

FFill Flood Fill

Function: Fills the area where the background is the same color as the draw pointer. Filling starts at the current draw pointer position, using the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 4F

Parameters: None

Line Draw Line

Function: Draws a line from the current draw pointer position to the specified point, using the current foreground color. The draw pointer returns to its original location.

Code: 1B 44

Parameters: HBX LBX HBY LBY

RLine Relative Draw Line

Function: Draws a line from the current draw pointer position to the point specified by the *x,y* offsets, using the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 45

Parameters: HBX_o LBX_o HBY_o LBY_o

LineM Draw Line and Move

Function: Draws a line from the current draw pointer position to the specified point, using the current foreground color. The draw pointer stays at the new location.

Code: 1B 46

Parameters: HBX LBX HBY LBY

RLineM Relative Draw Line and Move

Function: Draws a line from the current draw pointer position to the point specified by the offsets, using the current foreground color. The draw pointer stays at the new location. This is a relative command.

Code: 1B 47

Parameters: HBX_o LBX_o HBY_o LBY_o

Point Draw Point

Function: Draws a pixel at the specified coordinates, using the current foreground color.

Code: 1B 42

Parameters: HBX LBX HBY LBY

RPoint Relative Draw Point

Function: Draws a pixel at the location specified by the offsets, using the current foreground color. This is a relative command.

Code: 1B 43

Parameters: HBX_o LBX_o HBY_o LBY_o

PutGC Put Graphics Cursor

Function: Puts and displays the graphics cursor at the specified location. The coordinates passed to this command are not window-relative. The horizontal range is 0 to 639. The vertical range is 0 to 191. The default position is 0,0.

This command is useful for applications running under GrfInt so that you can display a graphics cursor under WindInt even if you don't want mouse control of the cursor.

Code: 1B 4E

Parameters: HBX LBX HBY LBY

SetDPtr Set Draw Pointer

Function: Sets the draw pointer to the specified coordinates. The new draw pointer position is used as the beginning point in the next draw command if other coordinates are not specified.

Code: 1B 40

Parameters: HBX LBX HBY LBY

RSetDPtr Relative Set Draw Pointer

Function: Sets the draw pointer to the point specified by the offsets. The new draw pointer position is used as the beginning point in the next draw command if other coordinates are not specified. This is a relative command.

Code: 1B 41

Parameters: HBX_o LBX_o HBY_o LBY_o

Text Commands

The text commands let you control the cursor's position and movement and also the way text prints on the display. These commands can be used on either text or graphics windows.

The text commands are:

Code	Description
01	Homes the cursor.
02 $x\ y$	Positions cursor to x,y . Specify coordinates as $(x + \$20)$ and $(y + \$20)$.
03	Erases the current line.
04	Erases from the current character to the end of the line.
05 20	Turns off the cursor.
05 21	Turns on the cursor.
06	Moves the cursor right one character.
07	Rings the bell.
08	Moves the cursor left one character.
09	Moves the cursor up one line.
0A	Moves the cursor down one line.
0B	Erases from the current character to the end of the screen.
0C	Erases the entire screen and homes the cursor.
0D	Sends a carriage return.
1F 20	Turns on reverse video.
1F 21	Turns off reverse video.
1F 22	Turns on underlining.
1F 23	Turns off underlining.
1F 24	Turns on blinking. ¹

Code	Description
1F 25	Turns off blinking. ¹
1F 30	Inserts a line at the current cursor position.
1F 31	Deletes the current line.
1B 3C BSW	See TCharSw in Chapter 3. ²
1B 3D BSW	See BoldSw in Chapter 3. ²
1B 3F BSW	See PropSw in Chapter 3. ²

¹ Blink is not supported for text on graphics screens.

² These characteristics are supported for text on graphics screens only.

Index

- active window 3-34
- AND 3-21
- arc, draw 4-2
- ARC3P 4-2

- background color 3-2, 3-13
- BAR 4-3
- bar, draw 4-3
- bar, relative draw 4-3
- BCOLOR 3-2
- bell, ring 5-1
- blinking 5-1, 5-2
 - off 5-2
 - on 5-1
- boldface 3-3, 5-2
- BOLDSW 3-3, 5-2
- BORDER 3-4, 3-26
- border color 3-4, 3-13, 3-26
- BOX 4-4
- box, draw 4-4
- box, relative draw 4-4
- buffer, kill 3-20
- buffer, load 3-19
- buffers 3-7, 3-19
- buffers
 - close 3-20
 - define 3-7
 - font 3-19
 - get/put 3-19
 - get/put 3-7
 - patterns 3-29
 - save 3-18
 - group numbers 3-7
 - kill 3-20
 - load 3-19
 - logic 3-21
 - pattern 3-29
 - 16-color 3-31
 - 2-color 3-30
 - 4-color 3-31
 - pattern array 3-29
 - pattern size 3-29
 - put 3-32
 - block 3-32
- buffers (*cont'd*)
 - screen type 3-32
 - size 3-32
 - save 3-18

- carriage return 5-1
- change font 3-15
- character
 - erase 5-1
 - transparent 3-35
- CIRCLE 4-5
- circle, draw 4-5
- close buffer 3-20
- close overlay window 3-23
- close, window 3-10, 3-23
- color 3-26
 - background 3-2, 3-13
 - border 3-4, 3-13, 3-26
 - composite 3-27
 - default 3-6
 - foreground 3-13, 3-14, 3-26
 - graphics 3-6
 - high-resolution 3-6
 - palette 3-26
 - RGB 3-26
 - VDG-emulation 3-6
- command parameters 2-1
- commands
 - drawing 2-1, 4-1
 - general 2-1, 3-1
 - text 2-1, 5-1
- composite colors 3-27
- create windows
 - device 1-2
 - overlay 1-4
- current window 3-13, 3-34
- cursor 3-17
 - home 5-1
 - graphics 3-17
 - put 4-11
 - move 5-1
 - off 5-1
 - on 5-1

- cursor (*cont'd*)
 - position 5-1
 - set 3-17
- CWAREA 3-5
- default color 3-6
- default windows 1-3
- DEFCOLOR 3-6
- define buffers 3-7
- define device windows 3-12
- delete line 5-2
- device descriptors 1-1, 3-12
- device windows 1-1
 - background color 3-13
 - border color 3-13
 - color
 - background 3-13
 - border 3-13
 - foreground 3-13
 - define 3-12
 - end 3-10
 - foreground color 3-13
 - keyboard 3-34
 - location 3-13
 - process window 3-13
 - protect 3-11
 - select 3-34
 - set 3-12
- DFNGPBUF 3-7
- DISPLAY 1-2, 2-1, 3-12
- draw pointer 4-1
 - relative set 4-12
 - set 4-12
- draw
 - arc 4-2
 - bar 4-3
 - bar, relative 4-3
 - box 4-4
 - box, relative 4-4
 - circle 4-5
 - ellipse 4-6
 - fill 4-7
 - flood fill 4-7
 - line 4-8
 - line and move 4-9
 - draw (*cont'd*)
 - line and move,
 - relative 4-9
 - line, relative 4-8
 - point 4-10
 - point, relative 4-10
 - pointer 4-1
 - drawing commands 4-1
 - DWEND 3-10
 - DWPROTSW 3-11
 - DWSET 1-1, 1-2, 3-12
 - ELLIPSE 4-6
 - ellipse, draw 4-6
 - end overlay window 3-23
 - end window 3-10, 3-23
 - erase character 5-1
 - erase line 5-1, 5-2
 - erase screen 5-1
 - erase to end of screen 5-1
 - escape sequence 1-2
 - FCOLOR 3-14
 - FFILL 4-7
 - fill, draw 4-7
 - flood fill, draw 4-7
 - FONT 3-15
 - font 3-12, 3-15, 3-19
 - bit map 3-15
 - boldface 3-3, 5-2
 - change 3-15
 - current 3-15
 - data 3-15
 - load 3-19
 - order 3-15
 - proportional 3-15, 3-28, 5-2
 - font bit map 3-15
 - font data 3-15
 - font load 3-19
 - font order 3-15
 - foreground color 3-13, 3-14, 3-26
 - GCSET 3-17

-
- general commands 2-1, 3-1
 - get/put buffers
 - close 3-20
 - define 3-7
 - font 3-19
 - group numbers 3-7
 - kill 3-20
 - load 3-19
 - logic 3-21
 - patterns 3-29
 - put 3-32
 - save 3-18
 - GETBLK 3-18
 - GPLOAD 3-9, 3-15, 3-19, 3-32
 - graphic patterns 3-29
 - graphics
 - boldface 3-3
 - colors 3-6
 - cursor 3-17, 4-11
 - put 4-11
 - set 3-17
 - font, proportional 3-28
 - transparent 3-35
 - group numbers 3-7, 3-8
 - Grp_Clip 3-8
 - Grp_Fnt 3-8
 - Grp_Pat2 3-8
 - Grp_Pat4 3-8
 - Grp_Pat6 3-8
 - Grp_Ptr 3-8
 - high-resolution, colors 3-6
 - home cursor 5-1
 - I\$ATTACH 3-10
 - immortal shell 1-3
 - INIZ 1-2, 3-10, 3-12
 - insert line 5-2
 - interactive window 3-34
 - KILBUF 3-20
 - kill buffer 3-20
 - LINE 4-8
 - line
 - delete 5-2
 - draw 4-8
 - erase 5-1, 5-2
 - insert 5-2
 - relative draw 4-8
 - line and move, draw 4-9
 - line and move, relative draw 4-9
 - LINEM 4-1, 4-9
 - load get/put 3-19
 - load font 3-15, 3-19
 - location, window 3-13
 - logic operations 3-21, 4-1
 - logic set 3-21
 - LSET 3-21
 - memory 1-1, 3-12
 - MONTYPE 3-6
 - move cursor 5-1
 - open windows
 - device 1-2
 - overlay 1-4
 - OR 3-21
 - overlay window 1-1, 1-2, 3-23, 3-24
 - overlay window
 - end 3-23
 - no-save 3-24
 - save 3-24
 - select 3-34
 - set 3-24
 - size 3-24
 - stacked 3-25
 - OWEND 3-23
 - OWSET 1-4, 3-24
 - PALETTE 3-26
 - palette colors 3-26
 - parameters, command 2-1
 - pattern 3-29, 4-1
 - 16-color 3-31
 - 2-color 3-30
-

- pattern (*cont'd*)
 - 4-color 3-31
 - array 3-29
 - size 3-29
- POINT 4-10
- point, draw 4-10
- point, relative draw 4-10
- pointer, draw 4-1
- position cursor 5-1
- process window 3-13
- proportional characters 3-28, 5-2
- proportional font 3-28
- PROPSW 3-28, 5-2
- protect device windows 3-11
- PSET 3-29
- put block 3-32
- put buffer 3-32
 - screen type 3-32
 - size 3-32
- put graphics cursor 4-11
- PUTBLK 3-18, 3-19, 3-32
- PUTGC 4-11

- RBAR 4-3
- RBOX 4-4
- reverse video 5-1
- RGB colors 3-26
- ring bell 5-1
- RLINE 4-8
- RLINEM 4-9
- RPOINT 4-10
- RSETDPTR 4-1, 4-12

- save, get/put 3-18
- save window 3-18
- SCALESW 3-33
- scaling 3-33
- scaling, automatic 3-33
- scaling coordinates 3-33
- screen type 3-12
- screen, erase 5-1
- SELECT 3-34
- select window 3-34

- set
 - device window 3-12
 - draw pointer 4-12
 - draw pointer, relative 4-12
 - overlay window 3-24
- SETDPTR 4-1, 4-12
- SETSTAT 3-6
- SHELL 1-3
- shell, fork 1-3
- stacked overlay windows 3-25

- TCHARSW 3-35, 5-2
- text commands 5-1
- text
 - boldface 3-3, 5-2
 - proportional 3-28, 5-2
- transparent character 3-35, 5-2

- underline
 - off 5-1
 - on 5-1

- video, reverse 5-1

- WCREATE 1-2
- windows 1-1
 - background color 3-2, 3-13
 - boldface 3-3, 5-2
 - border color 3-4, 3-13, 3-26
 - buffers 3-7, 3-19
 - kill 3-20
 - load 3-19
 - patterns 3-29
 - put 3-32
 - close 3-10, 3-23
 - color 3-26
 - background 3-2, 3-13
 - border 3-4, 3-13, 3-26
 - composite 3-27

-
- windows (*cont'd*)
- default 3-6
 - foreground 3-13, 3-14, 3-26
 - RGB 3-26
 - current 3-13
 - cursor 3-17
 - default 1-3
 - default color 3-6
 - device 1-1
 - define 3-12
 - end 3-10
 - opening 1-2
 - protect 3-11
 - select 3-34
 - set 3-12
 - device descriptors 1-1, 3-12
 - end 3-10
 - fonts 3-15, 3-19
 - foreground color 3-13, 3-14, 3-26
 - graphics cursor 3-17, 4-11
 - interactive 3-34
 - keyboard 3-34
 - location 3-13
 - logic operations 3-21
 - maximum 1-1
- windows (*cont'd*)
- memory 1-1, 3-12
 - overlay 1-1, 1-2, 3-23, 3-24
 - end 3-23
 - no-save 3-24
 - opening 1-4
 - save 3-24
 - select 3-34
 - set 3-24
 - size 3-24
 - stacked 3-25
 - process window 3-13
 - process 3-13
 - protect 3-11
 - put buffer 3-32
 - save 3-18
 - scaling 3-33
 - screen type 3-12
 - select 3-34
 - size 3-5
 - transparent mode 3-35, 5-2
 - type 1-1
 - work area 3-5
- work area 3-5
- XOR 3-21

OS-9
Glossary

OS-9 Glossary

active processes. Operations that the system is currently executing.

active state. An operating or working condition. A procedure in an active state is processing data and not waiting for another procedure to end.

address. A number that identifies a location in your computer's memory.

age. A count of the number of switches (process changes) the system has made since a process's last time slice.

anonymous directory. A directory referenced by its hierarchical position using the period (.) character. One period refers to the current directory. Two periods refer to the parent of the current directory, and so on.

application program. A process or group of processes designed to accomplish specific tasks, such as word processing, data management, game playing, and so on.

argument. Data you supply to a process or command for it to evaluate.

array. Data arranged so that each item is located by its row and column position. Single-dimensional arrays have one or more rows and one column. Multi-dimensional arrays have one or more rows and two or more columns.

ASCII code. American Standard Code for Information Interchange. A method of defining alphabetic and numeric characters and other symbols by giving each a unique value. For instance, the ASCII value for A is 65, and the ASCII value for B is 66.

assembler. A program that produces machine code from source code (code from a low-level computer language).

assembly language. A system for coding computer instructions to perform tasks. You can use assembly language code to directly manipulate data within a computer; therefore, assembly language needs less interpretation than higher level languages like BASIC or Pascal.

attribute. *See* file attribute.

background processing. Executing one or more procedures and at the same time continuing to operate in OS-9 or in another procedure.

backup. An identical copy of the contents of one disk on another disk.

base. The lowest value allowed in a function or operation. For instance, BASIC09 allows a base value of 1 for array structures unless you indicate otherwise.

batch file. See procedure file.

baud. Bits-per-second. A unit for measuring the speed of data flow between devices.

binary. A numbering system using only two digits, 0 and 1. In this system, shifting the position of a digit to the left raises the value of the digit by the power of 2. For instance, 1 is the binary equivalent of 1, $10 = 2$, $100 = 4$, $1000 = 16$, and so on.

bit. The smallest unit of a computer's memory. Eight bits form a byte. Each bit can have a value of either 0 or 1.

bit map. A storage area of 256 bytes. Each bit represents one page (256 bytes) of your computer's memory. If a bit is set (equals 1) then its associated memory page is allocated. If a bit is reset (equals 0) then its associated memory page is free.

block. A group of data, often comprising 256 bytes.

block-oriented device. A device that receives data, sends data, or both, in groups of 256 bytes.

Boolean logic. A binary type of algebra developed by George Boole.

Boolean data type. A type of variable that can have only two values, True or False. Boolean data types usually store the results of comparisons, such as: is A greater than B ($A > B$), does Y equal X ($Y = X$), and so on.

boot. The process of loading and initializing OS-9.

bootfile. A disk file containing modules to be loaded during an OS-9 boot.

bootlist. A disk file containing a list of module names to be used by OS9Gen to create a bootfile.

bootstrap module. A program that contains the code necessary to initialize OS-9.

border. An area around a screen or window that defines the boundaries of the screen or window.

branch. To leave one routine and begin execution of another routine within a program or procedure.

breakpoints. Locations in a program or procedure at which you want execution to pause.

buffer. A temporary storage area through which OS-9 transfers data.

byte. A unit of computer memory storage that contains a value in the range 0-255.

byte data type. A numeric type of variable that can contain unsigned eight-bit integer data (in the range 0-255 decimal).

call. (1) To transfer execution to another routine, then return to the calling procedure with obtained values intact and available for use by the calling routine. (2) A built-in OS-9 routine that performs a system function.

CC3Disk. The floppy diskette driver module.

CC3IO. The system input/output driver.

chaining. A process of calling and turning over system control to a new procedure.

checksum. A value calculated from the contents of a file or module that the system can later use to verify whether the contents of the file or module are uncorrupted.

child or child process. A process begun from another (parent) process.

close. The process of deallocating the path to a device or file.

cluster. A group of sectors. In OS-9 for the Color Computer, a cluster consists of only one sector.

code. Numeric data that can be used by a computer to perform a task.

command. The name of an OS-9 program or function.

command line. One or more commands with all their parameters, options, and modifiers.

command modifiers. Data or values appended to a command that change the way the command functions.

command options. Data that you can include in a command line to specify the way the command functions.

command parameters. Data or values appended to a command that define or customize the command.

command separator. A semicolon. You can use a semicolon to separate several commands on the same command line.

compile. To create machine language code (object code) from a program written with a computer language. Also to translate high level code (from a high level language such as BASIC) into low level code (code that is like machine language).

complement. A value that is derived by subtracting a number from a constant. For example, the 10s complement of 4 is 6. In binary, a value is complemented by changing all the 1 digits to 0 and all the 0 digits to 1, then adding 1 to the least significant (rightmost) digit.

complex data structure. A group of data that contains two or more types of data structures. *See* data structure.

constant. A value or block of data that is fixed (does not change during the run of a program or procedure).

CPU. Central Processing Unit. An integrated circuit (chip) that controls the operation of a computer.

current directory. The directory in which OS-9 looks for data files or stores data files unless you specify otherwise.

current line. When editing, the line on which the editing cursor or pointer is located.

cursor (text). A colored box that shows where the next character is to appear on the screen. A text cursor appears on both text and graphics windows or screens.

cyclic redundancy check (CRC). A value the system calculates from the data stored in a module. The system calculates a new value each time it attempts to load the module. If the calculated value does not match the CRC value contained in the module, the system cannot load the module.

cylinder. A disk track that includes both sides of a disk. *See also* track.

DAT. Dynamic address translation. The memory management system used by OS-9 Level Two.

data directory. The directory in which OS-9 automatically saves files unless you specify otherwise.

data structure. A unit of data, organized for access.

data type. A method for representing data, such as character (ASCII value), integer (whole number), or real (floating point number).

deadlock. *See* deadly embrace.

deadly embrace. A situation in which two processes attempt to gain control of the same disk areas at the same time.

debug. To find and correct program errors.

decompile. To translate machine language code into a computer language code.

delimiter. A character that divides items. For instance, in OS-9, the semicolon is a delimiter that divides two commands on the same line.

descriptor. *See* device descriptors.

device. A data source, destination, or both. OS-9 devices can exist in your computer's memory (such as a window or a RAM disk), or they can be external equipment (such as a printer or disk drive).

device descriptors. Small tables that define a device, its name, its driver, and its file manager. Device descriptors also contain port initialization data and port address information.

device drivers. Modules that handle basic input/output functions for specific devices. Each device you use with your computer must have its own driver to interpret the code you send it.

device name. A unique system word for a device. The name for disk Drive 0 is /D0, the name for Window 1 is /W1, and so on.

device table. *See* device descriptor.

device window. An OS-9 device from which you can run a program or utility. Access device windows in the same manner as you do other devices. Each device window has its own input and output buffers. Refer to windows using device names (/W followed by a number), such as /W1, /W2, /W3, and so on.

directory. A file in which OS-9 stores a list of other files, including their names, locations on the disk, attributes, and so on.

disk allocation map. Logical Sector Number 1 on a disk. The data in LSN1 indicates which sectors are allocated to files and which sectors are free.

double click. To press and release the mouse button twice in quick succession when the pointer is over the desired location.

drag. To hold down the mouse button and move the mouse to a new position before releasing the button.

draw pointer. An indicator that determines where the next graphics draw command will begin unless you specify otherwise.

driver. *See* device driver.

dump. To write the contents of a video screen, a memory location or a file to another terminal, memory location, or file.

echo. To cause data being sent to one device to go to another device, as well.

edit. To change the data or values in a file or in your computer's memory.

edit buffer. An alternate workspace for the OS-9 Macro Editor.

edit macro. A series of commands you can execute with only a single command.

edit pointer. An indicator that determines where the next edit command is to operate unless you specify otherwise.

editor. A program that provides special commands to aid you in changing the contents of a file.

error code. A code that OS-9 displays when it cannot understand what you want it to do or when your computer or a peripheral malfunctions. Use the displayed code number to look up a description of the error.

error path. The route through which OS-9 sends error codes and other information to display on the screen. The error path is designated as Path Number 2.

error trap. A routine in a procedure that checks for an error and provides an alternate action (other than terminating execution and displaying a system error message).

executable file. A program file that you can run by typing and entering its filename.

execute. To start a procedure, program, or command (cause it to run).

execution directory. The disk directory that contains your system's command files.

execution modifiers. *See* command modifiers.

execution offset. The location in a program or subroutine at which execution begins, calculated from the beginning of the module.

expressions. Data items joined by arithmetic operators. *See also* operator.

expression stack. A memory location in which BASIC09 stores temporary results while it evaluates an expression.

file. (1) A block of information your computer uses for a particular function or program. A file can contain an operating system, a language, an application program, or text. (2) A collection of associated records, such as information about each book in a library.

file attribute. Data that identifies a file, for instance its size, security status, language type, and so on.

file locking. Protecting a file to ensure that one process does not change it while another process is using it.

file pointer. An indicator that determines where in a file the next read or write operation is to occur unless you or the system indicates otherwise.

file security. A set of attributes that determines who can use a file and in what manner.

filename. A set of characters that uniquely identifies and locates a block of data stored on a disk.

filter. To alter data in some manner as it passes between two devices or between two memory locations.

flag. A symbol or value that indicates when a certain condition exists in a procedure.

font. A character set. A group of alphabetic and numeric characters and other symbols of a particular style and shape.

foreground. (1) An OS-9 task that takes priority over other concurrently running tasks. (2) Characters or designs on a screen or window.

fork. The process of initializing one procedure from another procedure.

format. To magnetically organize a diskette so that the computer can use it to store data.

function. In BASIC09, an operation that BASIC performs on data. A function always returns (produces) a value of some type.

Get/Put buffer. A buffer in which you or the system can store fonts, screen patterns, graphic displays, overlay windows, and other recallable data. The system allocates Get/Put buffers in 8-kilobyte blocks.

getstat. An OS-9 routine that gets (returns) the state or status of a specific system operation.

global variable. A variable that is available to all procedures and routines in a program.

graphics. An arrangement of elements (lines, dots, and so on) on your computer's screen.

graphics cursor. An indicator (either visible or invisible) that determines where the next graphic function is to occur on the screen unless you or a program specifies otherwise. In applications, you often move the graphics cursor using a mouse.

graphics pointer. *See* graphics cursor.

graphics screen. A screen in which all pixels are represented by bits in a memory map. You create images on the screen by manipulating the bits using special OS-9 or computer language commands.

graphics window. A window created on a graphics screen. You can display both graphics (drawn images) and text on a graphics window. The text generated on a graphics window/screen uses software fonts that you or the system must load into memory.

group. An organization of related data or files. For instance, OS-9's graphics buffers are organized into groups that you reference by number.

hardware. The physical parts of your computer, including its disk drives, keyboard, integrated circuits (chips), and so on.

header. Data located at the beginning of a file or module to identify its type, size, verification values, and so on.

hexadecimal. A number system to a base of 16 (using 16 digits). Hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Shifting a hexadecimal digit one place to the left causes its value to be multiplied by 16.

high order bit. The most significant or leftmost bit in a byte. If the high order bit is 0, it represents a value of 0. If the high order bit is 1, it represents a value of 128.

I/O. Input/Output.

identification sector. Logical Sector Number 0 on a disk. LSN0 contains a description of the physical and logical organization of a disk.

immortal shell. An OS-9 shell that does not die on receiving an EOF signal (such as when you press `CTRL BREAK`).

integer data type. A type of variable that can store whole numbers in the range -32768 to 32767.

interactive window. A window that is getting input from the keyboard. This window is currently on the displayed screen.

interface. To link devices or modules together in order to transfer data.

internal integrity check. A system of internal values that OS-9 can use to make certain that its system modules and functions are accurate.

IOMAN. The input/output manager that provides common processing for all input/output operations.

IRQ. Interrupt request. A signal that causes the execution of one process to halt and the execution of another process to begin. The system retains the values of the first process so that it can later continue its execution.

kernel. OS-9 software that supervises the OS-9 system and provides basic system services, such as multitasking and memory management, and that links all system modules.

key sequence. Two or more keys you press at the same time to produce a specific function.

keyboard mouse. An OS-9 function that lets you use the keyboard arrow keys instead of an external mouse device. Press **CTRL CLEAR** to toggle the keyboard mouse on and off.

keyword. A command name.

kill. Terminate the execution of a process.

kilobyte. 1024 bytes.

link. To make a module available to a process.

link count. The number of processes using a module. When a module's link count reaches 0, OS-9 deallocates the module.

load. To transfer data from an external device into your computer's memory.

local variable. A variable that can be used by only the procedure or routine in which it resides.

locked. See file locking.

lockout. See file locking.

log in. To initiate the necessary procedure to operate OS-9 from a separate terminal (type in a user name and password).

logical address. An offset address. An address that is numbered from the beginning of a block rather than from the beginning of memory, a module, or other storage area.

logical sectors. Sectors that OS-9 or a program references in numeric order, regardless of their actual physical location on a disk.

loop. A sequence of BASIC09 commands that execute repeatedly a specified number of times or until a specific condition occurs to terminate the execution.

macro. A series of commands you can execute with only a single command name.

map. *See* memory map.

mask. A pattern of bits that you use in combination with a logical operator to change specified data selectively — reversing certain bits without affecting the others.

megabyte. One million bytes.

memory. The portion of your computer that stores data and values.

memory management. Assigning and mapping memory to keep track of modules (processes and data) and their uses.

memory map. A chart depicting the use of your computer's memory by the operating system.

menu. A screen display from which you select an action for your computer.

microprocessor. An integrated circuit (chip) that controls the basic operation of your computer.

mode. A particular function of a program or system.

modem. Modulator/demodulator. A device to prepare signals for transmission through telephone lines and to reverse the process after transmission.

modifier. *See* command modifier.

module. An OS-9 program or block of data residing in your computer's memory.

module body. A module's code (program or data), including the module name.

module directory. A table in your computer's memory that lists all the modules residing in memory.

module header. Code that resides at the beginning of all modules and that contains information about the module, including size, type, attributes, storage requirements, and its execution starting address.

monitor. The video display device connected to your computer.

mouse. A device you use to control a pointer on the display screen. In application programs, you can often use a mouse to indicate functions you want to initiate.

multi-programming. A method of computer operation in which the system allocates *slices* of execution time to more than one process in order to execute them concurrently.

multi-tasking. Executing more than one process at the same time.

multi-user. A system that lets more than one person access its functions at the same time.

nesting. Incorporating one structure into another structure of the same type. Both procedures then retain their individual identities.

non-shareable file. A file or module that can be used by only one procedure or user at one time.

null string. A string variable that does not contain a value (has a length of 0 characters).

object code. Machine language instructions.

offset. The difference between a location and a beginning location. For instance, you can tell some BASIC09 graphics functions to begin operation at a location that is offset from the current draw pointer position.

operand. A value that is used or manipulated during an operation or during the execution of an instruction.

operating system. A set of associated programs that carry out your commands.

operator. A symbol or word that signifies some action to be performed on specified data.

options. *See* command options.

output path. The route through which the system sends data from one device to another.

overflow. A condition in which a storage space is not large enough to contain the data sent to it.

overlay. A condition in which programs or modules in a computer's memory are replaced with other data.

overlay window. A window opened or placed on top of a device window.

overwrite. To replace data with other data.

owner. An entity that has control over a file, module, or process.

pack. To compile a BASIC09 procedure. *See* compile.

padding. Adding spaces to a string or unit of data to make it a specific length.

page. In your computer's memory, a block of 256 bytes.

paint. To fill all or a portion of the screen with a color.

palette. A register that contains a numeric code representing a color or shade.

parameters. *See* command parameters.

parent or parent process. A process that forks (starts) another process (a child process).

parity. A system in which all binary numbers of a code are converted to either even-bit numbers (an even number of 1s) or odd-bit numbers (an odd number of 1s).

parse. To search through a list or sequence of data.

Pascal. A high-level computer language.

pass by value. When BASIC09 passes a value from one procedure to another by evaluating a constant or expression and placing the result in temporary storage to be accessed by the second procedure.

pass by reference. When BASIC09 passes a variable from one procedure to another by providing the second procedure with the address of the variable's storage.

passive window. Any window that is not receiving input from the keyboard. A *process* can be running on a passive window provided that the process is getting its input from a source other than the keyboard.

pathlist. The route from one position in a disk's directory to another directory or file.

peripherals. Devices connected to your computer, such as printers, disk drives, and so on.

permission. The attributes of a file or module that determine who can use the file or module and in what manner.

physical sectors. The actual arrangement of sectors on a disk's surface, regardless of any internal organization by OS-9.

pipe. A function in which the output of one process becomes the input of another process.

pipeline. A series of commands, each of which passes the results of its operations to the next command in the series.

PIPEMAN. The pipe file manager. Pipes are memory buffers acting as files to transfer data between processes.

pixel. The smallest area of a display screen that can be manipulated (turned off or on).

pointer. An indicator that determines a location in memory, in a file, or on the screen.

port. A junction between devices through which data flows. An electrical connection between your computer and a peripheral.

position independent module. A module that need not be loaded at any certain location in memory.

procedure. A program or routine your computer can execute.

procedure file. A file containing one or more OS-9 commands. You can execute a procedure file in the same manner as you execute OS-9 commands or programs.

process. A computer program or a routine that performs a specific task as part of a computer program.

process descriptor. A block of data that includes information about a process, its state, memory allocations, priority, queue pointers, and so on.

process ID. A unique number the system gives each process it executes.

process priority. A value you or the system gives to a process that determines the amount of CPU (execution) time it is to receive in a multi-tasking environment.

process state. The condition of a process in regard to its execution. A process can be active (executing), waiting (awaiting its turn for processing), or sleeping (inactive until it receives a signal to awaken).

program. Code that causes your computer to perform some function or a series of functions.

program modules. Executable code. Modules you can run to perform a function or series of functions.

public. Any user of a program or module other than the owner. *See* owner.

purge. Delete. Usually refers to removing all, or a selected group, of files from a directory.

RAM. Random access memory. Computer memory you can write to (change) and read from.

RAM disk. A portion of your computer's memory that OS-9 can use for data storage and retrieval in the same manner as it uses an external disk drive. However, be certain you copy RAM disk data to a floppy diskette or hard disk before you exit OS-9 or turn off your computer. If you do not, the data is lost.

random access. Reading (accessing) information in a block of data without first having to read any preceding data.

raw data. Unformatted information that is passed to a device exactly as it exists.

RBF. The random block file manager that processes all disk input/output.

re-entrant programs. Programs or modules that can be used by more than one process at the same time.

read. The process of transferring data from a device into the computer's memory.

read permission. System permission to read (withdraw data from) a file.

real data type. A type of variable that can store floating point numbers in the range $\pm 1 \times 10^{\pm 38}$

record. A collection of related data items that a program or process considers to be a unit for the purpose of processing. A subdivision of a file, such as all information about a single item in an inventory file.

record locking. Protecting a portion of a file to ensure that one process does not change it while another process is using it.

recursive procedure or routine. A procedure or routine that repeatedly executes itself (that contains a statement causing it to run itself one or more times).

register. A location within a computer's memory (often in the CPU) for storing values during arithmetic, logic, or transfer operations.

remarks. Text contained in a program that describes the program itself and that is not to be executed.

ROM. Read only memory. Computer memory containing constant values that the computer can read but cannot change.

ROOT directory. The parent directory of all files and directories on a disk. The ROOT directory is created by FORMAT.

run. To execute, or to cause a program or procedure to start.

runtime. The duration of a program's execution.

SCF. The sequential character file manager that handles non-disk input/output operations to devices such as printers and terminals.

scratched. Destroyed. When you copy one file over another file, or the contents of one disk onto another disk, any data existing in the second file or on the second disk is scratched.

sector. A division of a disk track. Disk tracks are organized into several sectors.

seek. To position a file pointer at a specific byte location in a file.

semigraphics. Graphics (designs on the display screen) using ASCII graphic characters.

sequential access. The process of reading data in order, one character at a time.

sequential execution. Executing a series of commands or processes, one after the other.

sequential file. A file consisting of records of various lengths that must be accessed one after the other, starting at the first record.

serial. Refers to transmissions in which data leaves or arrives at a location or device, with data units following one after the other in space or time.

Setstat. An OS-9 routine that sets (changes) the state or status of a specific system operation.

shell. The command interpreter.

sibling. One of two or more processes executed by the same parent process.

sign bit. The leftmost bit of a binary number that serves as an indicator to show whether the number is positive or negative. Normally, a value of 0 indicates positive, and a 1 indicates negative.

signal. An interrupt from the system or another process that changes a procedure's or a device's state. For example, signals set an active process to a waiting state, awaken an inactive or sleeping process, or change the display window.

single step. A procedure in the Debug mode that lets you execute one procedure statement and (optionally) view the results.

single-user file. A file that only one person can access at a time.

single-user module. A program that only one person can use at a time.

sleeping state. A situation where you or the system suspends a process for a specified time or until you or the system sends it a wakeup signal.

source code. Program code produced using a computer language. Before it can control a computer, source code must be translated into machine language, either by a compiler or a translator program. *See also* compiler.

stack. A storage area in your computer's memory in which data can be placed or recovered in sequence, from one end only.

standard error path. The route through which your computer sends error codes and other messages to the screen.

standard input path. The route through which you can send data to your computer (usually the keyboard).

standard output path. The route your computer uses to send data to the screen.

start up. To turn on your computer and initialize OS-9.

stop bits. One or two bits that a terminal program sends after each unit of data to indicate that the transmission of the unit is complete.

string. A group of alphanumeric characters.

string data type. A type of variable that can contain one or more ASCII values (values representing alphanumeric characters or other symbols). String data types can be any length, up to the capacity of your computer's memory.

structured programming. Building a program out of a series of procedures, each of which performs a specific task but combines with its associated procedures as one program.

subdirectory. A directory that resides within another (parent) directory.

subroutine. An operation that performs a specific task as part of a larger operation.

super user. The system user who has control of the entire system and access to all system files and modules. User Number 0.

symbolic debugging. An error correcting system that lets you pause program execution and view the current values of variables, using their program names.

syntax. The rules for forming legal instructions for your computer.

system. (1) A group of files and programs that provide you with control over your computer. (2) Your computer with all its attached devices.

system call. Built-in OS-9 routines that perform particular functions, such as accessing disk files, printing data on the screen, and so on.

table. A storage area in memory or on disk containing ordered data to be used by a process or function.

task. A unit of work performed by your computer.

terminal. A computer or an electronic device, with a screen and keyboard, connected to your Color Computer 3. You can access OS-9 functions from a terminal in the same manner as you can access them from your Color Computer 3 keyboard.

text files. Files containing printable characters, or the code representing such characters.

text screen. A Type 1 or 2 screen. Text screens use hardware generation of characters (fonts are not definable) and are often referred to as hardware screens or windows. Text screens cannot display graphics. Text operations occur faster in text windows/screens than on graphic windows/screens.

text window. Any window created on a hardware text screen.

time slice. The period of time between system clock ticks. A tick occurs every 1/60th of a second.

timesharing. A situation in which more than one person uses the same operating system.

token. In the BASIC language, a numeric value that represents a keyword.

trace. To display each procedure statement as it executes and view its results.

tracks. Magnetically created concentric circles created on a disk for the storage of data. Tracks are established when you format a disk.

transparent characters. Characters that display over screen images without erasing any of the area surrounding the characters.

unlink. To remove a module (program) from your computer's memory.

update mode. The condition of a file when it is open for both reading and writing.

user ID. A number that identifies the operator to which a process belongs.

user number. *See* user ID.

utility. A short program that performs a frequently required task, usually for the maintenance of your computer system or files.

variable. A unit of storage with no fixed value. You define a variable and locate it in your computer's memory using a variable name.

VDG. Video Display Graphics.

vector. A graphics line or portion of a line.

verify. To check data for accuracy.

wait state. A situation in which a process remains suspended until one of its child processes terminates or until it receives a wakeup signal from the system.

wake up. To continue the execution of a process that has been suspended.

wild card. A symbol that represents or takes the place of one or more other characters or symbols.

WINDINT. Window interface.

window. All or a portion of your video screen with specific formats (columns, lines, size, colors, and so on) and type (graphics, text, or both). An area of a screen in which you can run a process or which can receive input.

word length. The number of bits to transmit as one unit.

workspace. A portion of your computer's memory that BASIC09 establishes for the storage and manipulation of procedures.

write. To transfer data from the computer's memory to a device.

write permit. System permission to change the data in a file.

write protect. A method of protecting a diskette so that your computer cannot change the data on it.

RADIO SHACK
A Division of Tandy Corporation
Fort Worth, Texas 76102