Copyright and publication information

This manual reflects version 4.2 of OS-9.
Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Corporation.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.
## Contents

### Chapter 1: Basic Commands and Functions

- Logging onto a Timesharing System ................................................................. 10
- Introduction to the Shell ..................................................................................... 11
  - Performing a Command Search ...................................................................... 12
- Using the Keyboard ............................................................................................. 12
  - Line Editing Control Keys .............................................................................. 12
  - Interrupt Keys ................................................................................................. 15
  - The Page Pause Feature ................................................................................ 16
- Standard OS-9 Utilities ....................................................................................... 16
  - Using the help Utility ...................................................................................... 17
  - Using free and mfree ....................................................................................... 17

### Chapter 2: The OS-9 File System

- OS-9 File Storage ................................................................................................. 20
  - The File Pointer ............................................................................................... 20
  - Text Files ........................................................................................................ 22
  - Executable Program Module Files ................................................................. 23
  - Random Access Data Files ............................................................................. 23
  - File Ownership ............................................................................................... 23
  - Attributes and the File Security System ......................................................... 24
  - Directory Attributes ....................................................................................... 25
- The OS-9 File System ........................................................................................ 26
- Current Directories .............................................................................................. 27
  - Directories and Single-User Systems ............................................................. 27
  - Directories and Multi-User Systems ............................................................... 27
  - The Home Directory ...................................................................................... 28
  - Directory Characteristics ............................................................................... 28
- Accessing Files and Directories: The Pathlist .................................................... 29
  - Full Pathlists ................................................................................................. 29
  - Relative Pathlists .......................................................................................... 30
- Basic File System Utilities .................................................................................. 32
Command History ................................................................................................... 106 
Error Reporting ...................................................................................................... 107 

Chapter 5: Making Files

The make Utility ..................................................................................................... 110 
Running the Make Utility ..................................................................................... 112 
Implicit Definitions .............................................................................................. 112 
Macro Recognition ............................................................................................. 113 
make Generated Command Lines ........................................................................ 115 
make Options ..................................................................................................... 116 
Examples Using the make Utility ............................................................................. 117 
Updating a Document ......................................................................................... 117 
Compiling C Programs ........................................................................................ 118 
Using Macros ..................................................................................................... 119 
Creating make .................................................................................................... 120 

Chapter 6: Performing Backups

Incremental Backups ............................................................................................. 122 
Making an Incremental Backup: The fsave Utility .................................................. 122 
fsave Options ................................................................................................ 123 
The fsave Procedure ...................................................................................... 124 
Example fsave Commands ............................................................................. 125 
Restoring Incremental Backups: The frestore Utility ............................................... 126 
restore Options ............................................................................................ 126 
The Interactive Restore Process ...................................................................... 128 
Example Command Lines ............................................................................... 132 
Incremental Backup Strategies ............................................................................. 132 
Daily Backup Strategy .................................................................................... 132 
Single Tape Backup Strategy .......................................................................... 133 
Use of Tapes or Disks .................................................................................... 134 
Using the tape Utility .............................................................................................. 135 
Backing Up the System Disk .................................................................................. 137 
Formatting a Disk ............................................................................................... 137 
Multiple Drive Format ..................................................................................... 138 
Single Drive Format ......................................................................................... 138 
Continuing the Formatting Process . ................................................................. 139 
The Backup Procedure .................................................................................... 140 
Multiple Drive Backup .................................................................................... 141 
Single Drive Backup ............................................................................................ 141
Chapter 7: OS-9 System Management

Setting Up the System Defaults:

the Init Module ................................................................. 144
Extension Modules ............................................................ 150
Changing System Modules ............................................. 150
Making Bootfiles ............................................................. 152
  Bootlist Files ................................................................. 152
  Bootfile Requirements .................................................. 152
  Making RBF Bootfile .................................................... 153
Using the RAM Disk .......................................................... 153
  Volatile RAM Disks ....................................................... 154
  Non-Volatile RAM Disks ................................................ 154
Making a Startup File ....................................................... 154
  Initializing Devices: init r0 h0 d0 t1 p1 .......................... 156
  Loading Utilities Into Memory: load -z=sys/loadfile ...... 158
  Loading the Default Device Descriptor: Load bootobjs/r0.dd ................................. 159
  Multi-user Systems: tsmon /t1 &................................... 159
Time Zones and the TZ Environment Variable .................. 160
  Time Zones ...................................................................... 160
System Shutdown Procedure ............................................ 161
Managing Processes in a Real-time Environment ............... 163
  Manipulating the Priority of a Process ......................... 163
  Changing the System’s Process Scheduling .................. 164
  Using System-State Processes and User-State Processes ................................. 165
Using the tmode and xmode Utilities ............................... 166
  Using the tmode Utility .................................................. 166
  Using the xmode Utility .................................................. 167
The termcap File Format .................................................. 167
  termcap Capabilities ..................................................... 169
  Example String Notations .............................................. 174
  Example termcap Entries ............................................. 174
Appendix A: ASCII Conversion Chart 
  ASCII Symbol Definitions ........................................... 176
This chapter contains information that can help you get started with the OS-9 operating system. It discusses the most frequently used system commands. The following sections are included:

- Logging onto a Timesharing System
- Introduction to the Shell
- Using the Keyboard
- Standard OS-9 Utilities
- Using the help Utility
- Using free and mfree

This manual contains specific information about how to use the OS-9 operating system. For general information about OS-9 and how to get it up and running, refer to the Getting Started with OS-9 manual.
Logging onto a Timesharing System

If you are using a single-user system, such as a personal computer, you may skip this section.

Logging onto a multi-user system applies to both “hardwired” and “dial-up” terminals. Until you press the <Return> key, idle terminals on multi-user systems do nothing but “beep”. Pressing the <Return> key starts the log-on utility called login. login maintains system security and starts each user with a personalized environment.

For more information about login and tsmon, refer to the Utilities Reference manual.

Once login is started, the system requests your user name and the password the system manager assigned to you. The system echoes your user name, but for security purposes your password is not echoed. You are given three attempts to enter a valid user name and password before the program quits.

The following is an example of the login procedure:

```
OS-9000/80386 V2.0  80486/PCAT 93/10/24 14:51:12
User Name: smith
Password: [not echoed]
Process #10 logged on  93/10/24 14:51:20
Welcome!
[1]$ 
```

Depending on how the system is set up, a system-wide message of the day (MOTD) may display on your screen. Typically, you will be set up inside your main working directory; this allows one or more initial programs to run automatically. To log off, simply press the <Escape> (end-of-file) key or type logout any time your main shell is active.
Introduction to the Shell

Every operating system has a command interpreter. A command interpreter is a translator between the command you type in and the commands the operating system understands and executes. The command interpreter in OS-9 is called the “shell”.

The shell provides many functions and options. Chapter 4, The Shell exclusively discusses the shell and its features. Refer to that chapter for more detailed information on the shell.

The shell is normally started as part of the system startup sequence on a single user system or after logging onto a timesharing system.

The shell functions in two ways:

1. It accepts interactive commands from your keyboard.
2. It reads a sequence of command lines from a special type of file called a “procedure file” or “script file”. The shell executes each command line in the procedure file as if the command lines had been typed in manually from the keyboard.

When the shell is ready for command input, it displays a $ prompt. This enables you to enter a command line, followed by a carriage return.

The first word of the command line is the name of a command. It may be in upper or lower case. The command may be the name of the following entities:

- an OS-9 utility
- an application program or programming language
- a procedure file

Most commands accept additional parameters or options and some may require them. These parameters or options provide the command and/or the shell with additional information, such as searching file names and directory names. Most options are preceded by a hyphen (-) character. Each parameter is separated by a space character.
Performing a Command Search

The shell follows a special searching sequence for locating the command in memory or on disk. The search sequence is as follows:

1. Current module directory
2. Alternate module directory
3. Subsequent module directory (as specified by the MDPATH environment variable)
4. The current execution directory
5. Subsequent execution directory (as specified by the PATH environment variable)

Once this sequence is complete, the current data directory is searched for procedure file by the given name. If it cannot find the command you specified, the error 000:216, “file not found” should be reported.

Below is an example of a simple shell command line:

$ list myfile

The name of the command is list. The file name myfile is passed to the list command as a parameter.

Using the Keyboard

Most input to OS-9, programming languages, and application programs is line oriented. This means that as you type, the characters are collected, but not sent to the program until you press the <Return> key. This gives you a chance to correct typing errors before they are sent to the program.

OS-9 has several line editing features. Each of these features uses control keys generated by pressing the <Control> key and some other character key simultaneously.

Line Editing Control Keys

The line editing control keys are listed in Table 1-1.
<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Control&gt;A</code></td>
<td>Repeat the previous input line. The last line entered is displayed again, but not executed. The cursor is positioned at the end of the line. You can either enter the line as it is or add more characters to it. You can also edit the line by backspacing and typing over old characters.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;B</code></td>
<td>Move the cursor one space to the left (non-destructive).</td>
</tr>
<tr>
<td><code>&lt;Control&gt;F</code></td>
<td>Move the cursor one space to the right if the cursor is not at the end of the line (non-destructive).</td>
</tr>
<tr>
<td><code>&lt;Control&gt;H</code></td>
<td>Backspace to erase previous characters. Most keyboards have a special <code>&lt;backspace&gt;</code> key that can be used directly without using the <code>&lt;Control&gt;</code> key.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;I</code></td>
<td>Insert mode toggle key; switch input to insert mode. This enables you to insert characters into an existing input line. Insert mode is terminated by entering <code>&lt;Control&gt;I</code> again, another control sequence, or a carriage return.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;K</code></td>
<td>Truncate the line from the current cursor position to the end-of-line. Reset the end-of-line position to the cursor position.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;L</code></td>
<td>Delete the word to the left of the cursor, shift left what is currently right of the cursor, and leave the cursor position on the first character of the deleted word.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;M</code></td>
<td>End-of-record. This has the same effect as a carriage return.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;P</code></td>
<td>Display the current input line again. This is mainly used for hardcopy terminals that cannot erase deleted characters.</td>
</tr>
<tr>
<td><code>&lt;Control&gt;Q</code></td>
<td>Resume the input and output previously stopped by <code>&lt;Control&gt;S</code>. The <code>&lt;Control&gt;Q</code> function is known as “XON”.</td>
</tr>
</tbody>
</table>
Table 1-1. Line Editing Control Keys (Continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Control&gt;R</td>
<td>Delete the word to the right of the cursor, shift left all text that is currently to the right of the deleted word. Leave the cursor at its original position.</td>
</tr>
<tr>
<td>&lt;Control&gt;S</td>
<td>Halt input and output until &lt;Control&gt;Q is entered. The &lt;Control&gt;S function is known as “XOFF”. This is a function used by many serial I/O devices, such as printers, to control output speed.</td>
</tr>
<tr>
<td>&lt;Control&gt;W</td>
<td>Temporarily halt output so you can read the screen before data scrolls off. Output resumes when any other key is pressed. Refer to the section on the page pause feature for more information.</td>
</tr>
<tr>
<td>&lt;Control&gt;X</td>
<td>Delete the current line.</td>
</tr>
<tr>
<td>&lt;Control&gt;Z</td>
<td>Move the cursor to the beginning of the current line (non-destructive).</td>
</tr>
<tr>
<td>ESCAPE or &lt;Control&gt;[</td>
<td>End-of-file. All OS-9 I/O devices, including terminals, are accessed as files. This key simulates the effect of reaching the end of a disk file.</td>
</tr>
</tbody>
</table>
Interrupt Keys

There are two important control keys called interrupt keys. These work differently from the line editing keys because you can use them at any time, not just when a program requests input. They are typically used to halt or alter a running program. These control keys are detailed in Table 1-2.

Table 1-2. Interrupt Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Control&gt;C</td>
<td>Send an interrupt signal to the most recent program.</td>
</tr>
<tr>
<td></td>
<td>This functions differently from program to program. If a program does not make specific interrupt provisions, it aborts the program. If a program has provisions for interrupts, &lt;Control&gt;C usually provides a way to stop the current function and return to a master menu or command mode. In the shell you can use &lt;Control&gt;C to convert the foreground program to a background program, if the program has not begun I/O to the terminal.</td>
</tr>
<tr>
<td>&lt;Control&gt;E</td>
<td>Send a program abort signal to the program presently running.</td>
</tr>
<tr>
<td></td>
<td>In most cases, this key prematurely aborts the current program and returns you to the shell.</td>
</tr>
</tbody>
</table>

The control keys above are the key assignments commonly used in most OS-9 systems. You can, however, change the correspondence between control keys and their functions, so your keys may be different. To redefine the function of control keys, use the tmode utility. This command enables you to customize OS-9 to the specific computer’s keyboard layout.

For more information about tmode, refer to Chapter 7, OS-9 System Management or the Utilities Reference manual.
The Page Pause Feature

The “page pause” feature stops the scrolling of output on the screen. This is helpful when long lines of output scroll too quickly to be read. OS-9 counts output lines until a full screen has been displayed. It then halts output until you press any key. This is repeated for each screen of output.

In addition, page pause counts a wrapped line as a single line. If the screen is displaying lines that wrap, you may set the page length to a number smaller than 24 so that the page pauses at the bottom of a screen full of information.

You can use tmode to turn this feature on and off, or to change the number of lines per screen. The tmode commands are shown in Figure 1-3.

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmode pause</td>
<td>Turn the page pause mode on.</td>
</tr>
<tr>
<td>tmode nopause</td>
<td>Turn the page pause mode off.</td>
</tr>
<tr>
<td>tmode page=n</td>
<td>Set the page length to &lt;n&gt; lines, where &lt;n&gt; is the number of lines.</td>
</tr>
</tbody>
</table>

Standard OS-9 Utilities

OS-9 provides over ninety standard utilities and built-in shell commands. Some of the most commonly used utilities are listed in Figure 1-4.

Refer to the Utilities Reference manual for a more detailed explanation of each utility.

<table>
<thead>
<tr>
<th>attr</th>
<th>backup</th>
<th>build</th>
<th>chd</th>
</tr>
</thead>
<tbody>
<tr>
<td>chx</td>
<td>copy</td>
<td>date</td>
<td>del</td>
</tr>
<tr>
<td>deldir</td>
<td>dir</td>
<td>dsave</td>
<td>echo</td>
</tr>
<tr>
<td>edt</td>
<td>format</td>
<td>free</td>
<td>help</td>
</tr>
<tr>
<td>kill</td>
<td>list</td>
<td>mkdir</td>
<td>merge</td>
</tr>
<tr>
<td>mfree</td>
<td>pd</td>
<td>pr</td>
<td>procs</td>
</tr>
<tr>
<td>rename</td>
<td>set</td>
<td>setime</td>
<td>shell</td>
</tr>
<tr>
<td>wait</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using the help Utility

The most important command to learn when beginning to use the OS-9 utilities is `help`. The `help` utility is an on-line quick reference. To use this utility, type `help`, a utility name, and a carriage return. The utility function, syntax, and available options are listed.

For example, if you cannot remember the function or syntax of the `backup` utility, you can type `help backup` after the `$` prompt:

```
$ help backup
Syntax: backup [<opts>] [<srcpath> <dstpath>]
Function: backup disks
Options:
  -b=<size> use larger buffer (default is 4k)
  -r don’t exit if read error occurs
  -v do not verify
```

The descriptions are short and precise and using this command is a quick way to find information without looking up the utility in the documentation.

Typing `help` by itself displays the syntax and use of the `help` utility.

This same information is also available by typing the utility name followed by a question mark (`-?`). Each utility has the `-?` option.

Using free and mfree

During the format procedure, a disk is divided into data blocks of a pre-defined number of bytes. When OS-9 stores a file, the file’s contents are stored in physically contiguous blocks. To find out how many blocks are available on the disk, use the `free` utility. This utility displays the amount of unused disk space in number of blocks and bytes. It also displays the disk name, its creation date, and the capacity of the device.
For example:

```
$ free /h1
"OS-9000/68030 Hobbes’ Disk" created on: Thu Sep  7 03:37:10 1989
Capacity: 208935 blocks, 102.019 Mbytes
Free: 10 blocks, 0 bytes
Largest Free Block: 3 blocks, 0 bytes
```

free uses a 4K buffer by default. To increase the buffer size, use the \(-b\) option. For example, to use a 10K buffer you could type the following command:

```
$ free -b=10
```

The equal sign (\(=\)) is optional. You may also type: free -b10.

mfree displays the address and size of unused memory available for allocation.

For example:

```
$ mfree
Current total free RAM: 1808.00 K-bytes
```

To learn more about unused memory, use the `\(-e\)` option with mfree.

For example:

```
$ mfree -e
```

<table>
<thead>
<tr>
<th>Minimum allocation size:</th>
<th>4.00 K-bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of memory segments:</td>
<td>7</td>
</tr>
<tr>
<td>Total RAM at startup:</td>
<td>3841.90 K-bytes</td>
</tr>
<tr>
<td>Current total free RAM:</td>
<td>1808.00 K-bytes</td>
</tr>
</tbody>
</table>

Free memory map:

<table>
<thead>
<tr>
<th>Segment Address</th>
<th>Size of Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$7E000</td>
<td>$1000 4.00 K-bytes</td>
</tr>
<tr>
<td>$8D000</td>
<td>$1000 4.00 K-bytes</td>
</tr>
<tr>
<td>$A3000</td>
<td>$1000 4.00 K-bytes</td>
</tr>
<tr>
<td>$B9000</td>
<td>$1000 4.00 K-bytes</td>
</tr>
<tr>
<td>$CC000</td>
<td>$1BE000 1784.00 K-bytes</td>
</tr>
<tr>
<td>$291000</td>
<td>$1000 4.00 K-bytes</td>
</tr>
<tr>
<td>$296000</td>
<td>$1000 4.00 K-bytes</td>
</tr>
</tbody>
</table>
This chapter contains a detailed explanation of the tree-structured file and directory system. The following sections are included:

- OS-9 File Storage
- The OS-9 File System
- Current Directories
- Accessing Files and Directories: The Pathlist
- Basic File System Utilities
OS-9 File Storage

All information stored on an OS-9 computer system is organized into a set of files and directories.

- A file contains data, text, or a program.
- A directory contains names and locations of a file and other subdirectories within it.

This storage structure is hierarchical; it enables you to organize your files by topic, work group, or any other method. When a file is created, its information is stored as an ordered sequence of bytes. These bytes are organized into blocks. A block is a pre-defined group of bytes, anywhere from 256 bytes to 32768 bytes in powers of two. For example, a block may be composed of 512 bytes; every 512 bytes are grouped together as a block.

During the format procedure, each block is marked as “unused” and the allocation map keeps track of each. If a block is in use, it is marked in the allocation map as “in use” at the beginning of each disk. When a block is marked in use, OS-9 moves to the next available set of contiguous blocks and continues storing the information. Each of these sets of blocks is called a “segment”. The size of the segment is determined by the number of contiguous blocks available. When a file is shortened or deleted, the previously used blocks are unmarked in the allocation map and become available for use by another file.

Within a text file, each byte contains one character. Data is written to a file in the order it is provided. Data is read from a file exactly as it is stored in the file.

The File Pointer

When a file is created or opened, a file pointer is also created and maintained for it. The file pointer holds the address of the next byte to write or read (Figure 2-1). As data in the file is read or written, the file pointer is automatically moved. Therefore, successive read or write operations transfer data sequentially (Figure 2-2).
You can use an OS-9 system call, `seek`, to directly access any part of a file by positioning the file pointer to any location in the file. The `seek` system call can be accessed with the C function, `_os_seek`.

For more information about `_os_seek`, refer to the *Ultra C Library Reference* manual.

**Figure 2-1. Pointer Example 1**

```
  1  2  3  4  5
  r  o  b  e  r
```

When creating or opening a file, the file pointer is positioned to read from or write to the first component.

**Figure 2-2. Pointer Example 2**

```
  1  2  3  4  5
  r  o  b  e  r
```

After reading or writing the first component of a file, the file pointer points to the second component.

**Figure 2-3. Pointer Example 3**

```
  1  2  3  4  5
  r  o  b  e  r
```
The file pointer points to the current end-of-file (Figure 2-3). Attempting another read operation causes an end-of-file error. Another write operation increases the size of the file.

Figure 2-4. Pointer Example 4

```
1 2 3 4 5 6
r o b e r l
```

The next write operation adds a new component to the file and moves the file pointer to the new end-of-file (Figure 2-4).

Reading up to the last byte of the file causes the next read operation to return an end-of-file status (Figure 2-3). Attempting to read past the end-of-file mark causes an error. To expand a file, simply write past the previous end of the file (Figure 2-4).

Because all OS-9 files have the same physical organization, you can generally use file manipulation utilities on any file, regardless of its logical use. The main logical types of files used by OS-9 are listed below and are explained in the following sections.

- text files
- executable program module files
- data files
- directories (Directories are the exception and are covered separately.)

Text Files

Text files contain variable length lines of ASCII characters. Each line is terminated by a carriage return (hex $0D). Text files typically contain documentation, procedure files, and program source code. You can create text files with any text editor or the build utility.
Chapter 2: The OS-9 File System

Executable Program Module Files

Executable program modules store programs generated by assemblers and compilers. Each file may contain one or more modules with standard OS-9 module format.

For more information about modules, refer to the OS-9 Technical Manual.

Random Access Data Files

A data file is created and used primarily by high-level languages such as C, Pascal, and BASIC. The file is organized as an ordered sequence of records of varying sizes. If each record has exactly the same length, its beginning address within the file can be computed to enable you to access records in any order. OS-9 does not directly deal with records other than providing the basic file manipulation functions high level languages that support random access records require.

File Ownership

When you create a file or directory, OS-9 automatically stores a “group.user” ID with it. The group.user ID is formed from your group number and your user number.

- **group number**: Enable people working on the same project or working in the same department to share a common group identification.
- **user number**: Identify a specific user.

The group.user ID determines file ownership. OS-9 users are divided into three classes, as described in Table 2-1.

Table 2-1. User Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Any user with the same group and user number as the person who created the file. The super-user group (0.x) is also considered the owner of the file.</td>
</tr>
<tr>
<td>group</td>
<td>any user with the same group number as the person who created the file</td>
</tr>
<tr>
<td>public</td>
<td>any person with a group ID differing from the person who created the file</td>
</tr>
</tbody>
</table>
On multi-user systems, the system manager generally assigns the group.user ID for each user. This number is stored in a special file called a password file.

A user with a group.user ID of 0.0 is referred to as a “super user”. A super user can access and manipulate any file or directory on the system regardless of the file’s ownership. On a multi-user system, the super user is generally the system manager, although other people such as group managers or project leaders can also be super users. On single-user systems, users have super user status by default.

Attributes and the File Security System

File use and security are based on file attributes. Each file has ten attributes. These attributes are displayed in a sixteen character listing.

The term “permission” is used when one of the ten possible attribute characters is set. Permission determines who can access a file or directory and how it can be used. If a permission is not valid for the file or directory being examined, a hyphen (-) is in its position.

Below is an attribute listing for a file in which all permissions are valid:

-o---ewr-ewr-ewr

By convention, attributes are read from right to left. They include those shown in Table 2-2.

Table 2-2. File Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner Read</td>
<td>r</td>
<td>The owner can read the file.</td>
</tr>
<tr>
<td>Owner Write</td>
<td>w</td>
<td>The owner can write to the file.</td>
</tr>
<tr>
<td>Owner Execute</td>
<td>e</td>
<td>The owner can execute the file.</td>
</tr>
<tr>
<td>Group Read</td>
<td>gr</td>
<td>The group can read the file.</td>
</tr>
<tr>
<td>Group Write</td>
<td>gw</td>
<td>The group can write to the file.</td>
</tr>
<tr>
<td>Group Execute</td>
<td>ge</td>
<td>The group can execute the file.</td>
</tr>
<tr>
<td>Public Read</td>
<td>pr</td>
<td>The public can read the file.</td>
</tr>
</tbody>
</table>
Chapter 2: The OS-9 File System

Directory Attributes

Directories have slightly different attributes from files. Instead of attributes for permission to execute files, directories have attributes for permission to search through directories for files. Below is an attribute listing for a directory in which all permissions are valid:

\[ \text{do---swr-swrswr} \]

By convention, directory attributes are also read from right to left. They are listed in Table 2-3.

Table 2-3. Directory Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner Read</td>
<td>r</td>
<td>The owner can read the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When off, this denies any access to the file.</td>
</tr>
<tr>
<td>Owner Write</td>
<td>w</td>
<td>The owner can write to the file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When off, this attribute can be used to protect files from accidentally being deleted or modified.</td>
</tr>
<tr>
<td>Owner Search</td>
<td>s</td>
<td>The owner can search the directory for files.</td>
</tr>
<tr>
<td>Group Read</td>
<td>gr</td>
<td>The group can read the file.</td>
</tr>
<tr>
<td>Group Write</td>
<td>gw</td>
<td>The group can write to the file.</td>
</tr>
<tr>
<td>Group Search</td>
<td>gs</td>
<td>The group can search the directory for files.</td>
</tr>
<tr>
<td>Public Read</td>
<td>pr</td>
<td>The public can read the file.</td>
</tr>
<tr>
<td>Public Write</td>
<td>pw</td>
<td>The public can write to the file.</td>
</tr>
<tr>
<td>Public Search</td>
<td>ps</td>
<td>The public can search the directory for files.</td>
</tr>
<tr>
<td>Exclusive Use</td>
<td>o</td>
<td>When set, only one user at a time can open the file.</td>
</tr>
<tr>
<td>Directory</td>
<td>d</td>
<td>When set, indicates a directory.</td>
</tr>
</tbody>
</table>
The OS-9 File System

OS-9 uses a tree-structured, or hierarchical, organization for its file system on mass storage devices such as disk systems. (Figure 2-5.) Each mass storage device has a master directory, called the root directory.

The root directory is created automatically when a new disk is formatted. It contains the names of the files and the subdirectories on the disk. Every file is listed in a directory by name and each file has a unique name within a directory.

An OS-9 directory can contain both files and subdirectories. Each subdirectory can contain more files and subdirectories. This enables you to embed subdirectories within other subdirectories. The only limit to this division is the amount of available disk space.

Figure 2-5. The File System

With the exception of the root directory, each file and directory in the system has a parent directory. A parent directory is the directory directly above the file or directory being discussed. For example in Figure 2-5, the parent directory of file2 is SUB-DIRECTORY1. Likewise, the parent directory of SUB-DIRECTORY1 is the root directory.
Current Directories

Two working directories are always associated with each user or process. These directories are called the current data directory and the current execution directory.

The following terms are important in the discussion of directories:

- **data directory**: the location in which text files are stored and created
- **execution directory**: the location in which executable files live

This current directory concept enables you to organize your files and keep them separate from other users on the system. The word “current” is used to reflect the idea that you can move through the tree structure of the OS-9 file system to a different directory; this different directory becomes your current data or execution directory.

Directories and Single-User Systems

On a single user system, OS-9 chooses the root directory of your system disk as the initial current data directory. The initial current execution directory is the **CMDS** directory. The **CMDS** directory is located in the root directory of the system disk.

Directories and Multi-User Systems

On a multi-user system, the current data and execution directories are established as part of the initial login sequence. When you log in, your initial directories are set up according to your password file entry. A password entry is established for each user on a multi-user system. This entry lists information such as the user’s password and current directories.

The execution directory on a multi-user system is usually the **CMDS** directory, which is shared with other users. **CMDS** contains OS-9 utilities and other executable files.
The Home Directory

On typical multi-user systems, all users have their own data directory. Through the `/H0/CMDS` environment variable, each user may also have a private execution directory to avoid conflict with other users.

The private data directory enables you to organize your files by project, function, or any other method without affecting other users’ files. The data directory specified in the password file entry is known as the home directory. When you first login to the system, your information is placed in this directory. This can also be accomplished by using the `chd` utility without parameters.

On single user systems, you may establish a home directory by setting the `HOME` environment variable.

For more information about:

- `chd`: Refer to the *Utilities Reference* manual.
- the `HOME` environment variable: Refer to *Chapter 4, The Shell*.

Directory Characteristics

Below are some important characteristics relating to directory files:

- Directories have similar ownership and attributes as other files. However, directories always have the `d` attribute set, and contain attributes that allow for file searching, while files have attributes for executing files.
- Each file name within a directory must be unique. For example, you cannot store two files named `trial` in the same directory. Files may have different names if they are stored in separate directories.
- All files are stored on the same device as the directory in which they are listed.
- You are limited to the number of files you can store in a directory by the amount of free disk space in your system.
Accessing Files and Directories: The Pathlist

You can access all files or directories in your current data directory by specifying the name of the file or directory after typing the correct command. In cases where only a file or directory name is given, OS-9 does not look outside your current data directory to find that particular file or directory.

If you want to access a file that does not live in your current data directory or run a program that does not live in your current execution directory, you must either change the current directory or specify a pathlist through the file system for OS-9 to follow.

There are two types of pathlists: full pathlists and relative pathlists. These are explained in the following sections.

Full Pathlists

A full pathlist starts at the root directory and follows the directory names in the list down the file structure to a specific file or directory. A full pathlist must begin with a slash character (/). (Slashes separate names within the pathlist.)

The following example is a full pathlist from the root directory, /d1, through two subdirectories, PASCAL and TESTS, to the file futureval.

/d1/pascal/tests/futureval

The example below specifies a path from the root directory, /h0, through the USR subdirectory, to the NICHOLLE subdirectory.

/h0/usr/nicholle

A full pathlist begins at the root directory regardless of where your current data directory is located. It lists each directory located between the root directory and a specific file or subdirectory.
Example

Suppose your data directory is RESEARCH. A full pathlist to current would then be /h0/WORK/current (as shown in Figure 2-6).

Figure 2-6. Full Pathlist Example

Relative Pathlists

A relative path starts at the current directory and proceeds up or down through the file structure to the specified file or directory. A relative pathlist does not begin with a slash (/). (Slashes separate names within a relative pathlist.)

When you use a relative pathlist and the desired destination requires moving up the directory tree, you can use special naming conventions to make moving around the pathlist easier.

- A single period (.) refers to the current directory.
- Two periods (..) refer to the current directory’s parent directory.
- Add a period for each higher directory level.

For example, to specify a directory two levels above the current directory, three periods are required. Four periods refer to a directory three levels above the current directory.

You can also use a Unix-style pathlist such as ..../.../

⚠️ A relative pathlist begins at your current directory regardless of its location in the overall file structure.
Example
Suppose your data directory is RESEARCH. A relative pathlist to current would then be \( \ldots/current \) (as shown in Figure 2-7).

Figure 2-7. Relative Pathlist Example

Using a relative pathlist name substitute does not change the directory’s name.

The following example is a relative pathlist that begins in your current directory and goes through the subdirectory DOC and LETTERS to the file jim.

\[ \text{DOC/LETTERS/jim} \]

The next pathlist goes up to the next directory above your current directory and then through the subdirectory CHAP to the file page.

\[ \ldots/CHAP/page \]

The next pathlist specifies a file within your current directory. No directories are searched other than the current directory.
Basic File System Utilities

Some OS-9 utility commands manipulate the file system. These utilities include `dir`, `chd`, `chx`, `pd`, `build`, `makdir`, `list`, `copy`, `dsave`, `del`, `deldir`, and `attr`. The given examples refer to an example file system (Figure 2-8).

Figure 2-8. Diagram of a Typical File System
The dir Utility

The dir utility displays the contents of a directory. Typing dir by itself displays the contents of your current data directory. For the following example, the current data directory is /h0. Typing dir, as shown in Figure 2-8, results in the following output:

```
$ dir
    directory of .  13:56:58
    C    CMDS    DEFS    IO    LIB
    MACROS  SYS  SYSMODS  USR  STARTUP
```

To look at directories other than your current data directory, you must either provide a pathlist to the desired directory or change your current data directory.

To display the contents of another directory without changing your current data directory, type dir and the pathlist to the directory. For example, if you are in the root directory and you want to see what is in the DEFS directory, type the following command:

```
dir defs
```

dir now displays the names of the file in the DEFS directory. The name defs is a relative pathlist. You can type dir defs because DEFS is in your current data directory. You can also use the full pathlist, dir /h0/defs and get the same result.

To display the contents of your current execution directory, type dir -x.

dir and Wildcards

Wildcards can be used with dir and with most other utilities. OS-9 recognizes two wildcards, as described in Figure 2-4.

Table 2-4. OS-9 Wildcards

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>An asterisk (*)</td>
<td>An asterisk replaces any number of letter(s), number(s), or special character(s). Consequently, an asterisk by itself expands to include all of the files in a given directory.</td>
</tr>
<tr>
<td>A question mark (?)</td>
<td>A question mark replaces a single letter, number, or special character.</td>
</tr>
</tbody>
</table>
For example, the command `dir *` lists the contents of all directories located in the current data directory. The command `dir /h0/cmds/d*` lists all files and directories in the CMDS directory beginning with the letter d. The command `dir prog_?` lists all files in your current directory having a file name with `prog_` followed by a single character.

For more information on using wildcards, refer to Chapter 4, The Shell.

dir Options

dir has several options are fully documented in the Utilities Reference manual. The `-e` and `-r` options are discussed in this section.

The `-e` Option

The `-e` option gives an extended directory listing. An extended directory listing displays all files within the specified directory with their attributes, sizes, and the sectors where the files are stored. The following example uses the file structure shown in Figure 2-8.

```
$ dir usr/bob -e
Directory of USR/Bob 12:30:27
Owner Last modified Attributes Block Bytecount Name
------- ------------- ---------------- ------ --------- -----
22.150 89/09/25 1057 --------------wr 12CB0 5744 letter
22.150 89/09/19 1057 d-------------wr 12CAF 15944 PROG
22.150 89/09/25 1103 d-------------wr 12C90 11113 TEXT
```

The `-r` Option

The `-r` option displays the contents of the specified directory and any files contained within its subdirectories. Typing `dir usr/usr1 -r` lists the following output:

```
Directory of . 12:30:15
PROG TEXT letter
Directory of PROG 12:30:15
funct main
Directory of TEXT 12:30:15
manual
```

You can use any dir option with another. Typing `dir -er` displays all files within the current data directory, all files within its subdirectories, and provides an extended listing of their attributes and sizes.
The chd and chx Utilities

The chd and chx utilities enable you to travel around the file system.

- chd changes the current data directory.
- chx changes the current execution directory.

Using chd

To change your current data directory, type chd followed by a full or relative pathlist. For example, if your current data directory is /h0 and you want your current data directory to be USR, type chd and the pathlist of USR.

- Using a relative pathlist, type the following command:
  
  chd usr

- Using a full pathlist, type the following command:
  
  chd /h0/usr

Your current data directory is now USR. When you type dir, you should see the contents of USR:

  directory of . 14:04:32
  USR1    USR2    USR3

To see which files are in the USR1 directory, type dir usr1. Change directories by typing chd usr1, then dir after the new prompt.

To return to your home directory, which in this case is /h0, type chd without including a pathlist. After changing the directory, dir displays the contents of /h0.

Using chx

The chx command enables you to redefine an existing directory as a personal execution directory. If you have programs you do not want other users to execute, you may find it useful to define a personal execution directory. To do this, type chx, followed by a full or relative pathlist to the directory. When using a relative pathlist with chx, note that the pathlist is relative to your current execution directory.
If your current data directory is **USR** and you want to change your current execution directory from **CMDS** to **USR2**, you can type the relative pathlist `chx ../usr/usr2` or the full pathlist `chx /h0/usr/usr2`. When you type a command after you have changed your current execution directory, OS-9 searches **USR2** instead of **CMDS**.

Typing `dir -x` displays the contents of your current execution directory, **USR2**:

```
Directory of .. 20:54:18
map  pics  new.c
```

**Navigating through Directory Trees**

You can use special naming conventions to move around the file system. Naming conventions are periods specifying the current directories and directories higher in the file structure. Consider the following example:

- `.` refers to the current directory
- `..` refers to the parent directory
- `...` refers to two directory levels higher

When used as the first name in a path, you can use these naming conventions with relative pathlists.

**Examples**

The following examples relate to the file structure in Figure 2-9. The examples assume your initial current data directory is **PROG**.

- The contents of **PROG** are displayed below. It is functionally the same command as `dir`.
  
  ```
  dir .
  directory of . 14:04:32
  funct  main
  
  dir ..
  directory of .. 14:05:58
  ```
  
  PROG  TEXT  letter
• The example below displays the contents of `TEXT` by specifying a path starting with the parent directory:

```
dir ../text
   directory of ../text 14:06:47
manual
```

• The following command changes the current data directory from `PROG` to `USR3`:

```
chd .../usr3
```

`USR3` is accessed from `PROG` using the relative path `.../usr3`.

**Figure 2-9. Accessing Directories Using a Relative Path**
You can use any number of periods (.) to access higher directories. One period is added for each level. An error is not returned if you specify a greater number of directory levels above your current data directory than actually exist. Instead, this indicates the root directory on your system.

For example, the command below displays the contents of the root directory:

```
dir ..........
```

This may be helpful if you are not sure how deep into the directory structure you have travelled.

- The following example changes your current data directory from `PROG` to `MACROS`:

```
chd ........../macros
```

### Using the `pd` Utility

The `pd` utility displays the complete pathlist from the root directory to your current data directory.

For example (if your current data directory is `USR2`):

```
pd
/h0/USR/USR2
```

While inside your current execution directory, type `pd -x` to display the pathlist to that directory.

### Creating New Directories

To create new directories, use the `mkdir` utility. For example, to create a directory called `MARKET`, type the following command:

```
mkdir MARKET
```

`MARKET` now is a new directory in your current directory.

If you want the new directory created somewhere other than in your current directory, you must specify a pathlist. For example, `mkdir /h0/usr/MARKET` creates the new directory in `USR`.
Rules for Constructing File Names

When creating files and directories, you must follow certain rules. A file name can contain from one to 43 upper and lower case letters, numbers, and special characters, as listed in Table 2-5. While the file name may begin with any of the following characters or digits, each file name must contain at least one letter or number. Within these limitations, a name can contain any combination of the following examples:

Table 2-5. Characters Allowed in File Names

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper case letter:</td>
<td>A - Z</td>
</tr>
<tr>
<td>Lower case letter:</td>
<td>a - z</td>
</tr>
<tr>
<td>Decimal digits:</td>
<td>0 - 9</td>
</tr>
<tr>
<td>Underscore:</td>
<td>_</td>
</tr>
<tr>
<td>Period:</td>
<td>.</td>
</tr>
<tr>
<td>Dollar sign:</td>
<td>$</td>
</tr>
</tbody>
</table>

File names must not contain spaces. Instead, use an underscore (_) or a period (.) should be used to improve the readability of file and directory names. OS-9 does not distinguish upper case from lower case letters. For example, the names FRED and fred are considered the same name.

Typically, directory names are in upper case and file names are in lower case. It is not required, but it helps to easily distinguish directories from files.
Table 2-6 lists are some examples of legal names:

### Table 2-6. Legal File Names

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw.data.2</td>
<td>project_review_backup</td>
</tr>
<tr>
<td>X6809</td>
<td>$SHIP.DIR</td>
</tr>
<tr>
<td>...c</td>
<td>12345</td>
</tr>
</tbody>
</table>

Table 2-7 lists are some examples of illegal names:

### Table 2-7. Illegal File Names

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max*min</td>
<td>* is not a legal character</td>
</tr>
<tr>
<td>open orders</td>
<td>name cannot contain a space</td>
</tr>
</tbody>
</table>

File names beginning with a period are not displayed by `dir` unless you use the `-a` option. This enables you to hide files within a directory.

### Creating Files

You can create files in many ways. Text files are generally created with the `build` utility, the `edt` utility, or the μMACS text editor. These file building tools are provided with the OS-9 package.

### Creating Short Text Files

Use the `build` utility to create short text files. To use `build`, type `build`, followed by the name of the file you want to create. `build` responds with a “?” prompt, which tells you that `build` is waiting for input. To terminate `build`, type a carriage return at the “?” prompt. Consider the following example:

```
$ build test
? Creating a text file is easy
? when you use the build utility,
? but you cannot edit files with build.
?
$
```

You cannot edit files with `build`. 
Editing Text Files

To create and edit text files, use the `edt` utility. `edt` is a line-oriented text editor with the capability to create and edit source files. To use `edt`, type `edt` and the desired pathlist. `edt` displays a question mark (?) prompt and waits for an edit command. If the file is found, `edt` performs the following tasks:

1. Opens the file.
2. Displays the last line of the file.
3. Displays the ? prompt.

Using µMACS

Most people prefer using µMACS to create and edit files. µMACS is a screen-oriented text editor for creating and modifying text files and programs. Through the use of multiple buffers, µMACS enables you to display different files or different portions of the same file on the same screen. In addition, extensive formatting commands enable you to complete the following tasks:

- Reformat paragraphs with new user-defined margins.
- Transpose characters.
- Capitalize words.
- Change words or sections into upper or lower case.

For more information about µMacs, refer to the Utilities Reference manual.

Examining File Attributes

When you create a file using `build` or µMACS, only the owner read and owner write permissions are set. When you create a directory, it initially has all the permissions set except the single user permission.

To examine file attributes, use the `attr` utility. To use this utility, type `attr`, followed by the name of a file. Consider the following example:

```
$ attr newtest
--------------wr
```
The file `newtest` has the permissions set for owner reading and owner writing. Access to this file by anyone other than the owner is denied.

Users with the same group/user ID as the person who created the file are considered owners. However, if the file is created by a group 0 user, only users in the super group can read, write, or execute the file.

If you use `attr` with a list of one or more attribute abbreviations, the file attributes are changed accordingly, provided you have the proper write permission to access the file. You do not need to list the attribute abbreviations in any particular order. The letter `n` preceding an attribute removes that permission.

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

```
$ attr newtest -pw -pr -ne -npe
```

The owner always has the right to delete a file, change the user privileges, etc. Users in the same group have the same permissions as the owner.

The directory attribute is somewhat different than the other attributes. It could be dangerous to be able to change directory files to normal files or a normal file to a directory. For this reason, you cannot use `attr` to turn the directory (`d`) attribute on; use `mkdir` to turn this attribute on. Furthermore, you can only use `attr` to turn the directory attribute off if the directory is empty.

### Listing Files

The `list` utility displays the contents of files. By default, `list` displays the lines of text on your terminal screen. To examine a file, type `list`, followed by the name of the file. For example:

```
$ list test
Creating a text file is easy
when you use the build utility,
but you cannot edit files with build.
$
```
It is important to remember that you cannot list a directory. If you type the command `list USR`, the following error message and error number will be returned:

```
list: can’t open "USR". Error# 000:214.
```

The above message means you cannot access `USR` because it is a directory.

`list` displays text files. All distributed files in `CMDS` are executable program module files. If you try to list the contents of a random access data file or an executable program module file, you see what appears to be random data displayed on your screen. This may also include unprintable characters--such as escape or delete--that could change your terminal’s operating parameters. If the operating characteristics of your terminal are affected, try turning the terminal off and on. If this does not re-initialize the terminal, consult your terminal operating manual.

### Copying Files

The `copy` utility makes a duplicate of a file. To copy a file, type `copy`, followed by the name of the file to be copied, followed by the name of the duplicate file. For example:

```
$ copy test newtest
```

If you `list` the file `newtest`, it is an exact copy of `test`.

The file you are copying, as well as the duplicate file, can be located in any directory; these do not have to be in your current data directory. For files located outside of your current data directory, use full or relative pathlists. The following example uses Figure 2-11. The first command copies the file `gee` in the `USR2` directory to a file named `new.info` in the `TEXT` directory:

```
copy /h0/usr/usr2/gee /h0/usr/usr1/text/new.info
```

Assuming your data directory is `USR`, the following commands have the same effect:

```
copy /h0/usr/usr2/gee usr1/text/new.info
```

```
copy usr2/gee usr1/text/new.info
```

`gee` is copied from `USR2/gee` to `USR1/TEXT/new.info` using the command `copy usr2/gee usr1/text/new.info`. 
Copying a File into an Existing File

If you try to copy the contents of one file into an existing file, you will receive the following message: Error #000:218 Tried to create a file that already exists. If you know the file exists but you want to overwrite it anyway, use the -r option. For example, the following command replaces the contents of green with the contents of fall.

```
$ copy fall green -r
```

When you list the contents of both files, it becomes obvious that they are identical.
Chapter 2: The OS-9 File System

Copying Multiple Files

At some point, you may want to copy more than one file at a time into another directory. By using the 
\[ -w=<\text{dir}> \]
on option of \texttt{copy}, you can copy more than one file with a single command. For example, if your current directory is \texttt{PROG} and you want to copy all of the files in \texttt{PROG} into the \texttt{TEXT} directory, you would type the following command line:

\[
\texttt{$ \ copy \ * \ -w=../text} \\
\]

This option prints the name of the file after each successful copy. If an error occurs, the prompt \texttt{continue (y/n)} is displayed.

Copying Large Files

If you have a large file, the copy procedure may be slow because the system has to perform multiple read and write statements from a small 4K buffer. To make the copy procedure faster when copying large files, use the 
\[ -b \]
on option to increase the buffer size. To use the 
\[ -b \]
on option, type \texttt{copy}, the original file name, the new file name, and 
\[ -b=<\text{num}>k. \]

For example, typing \texttt{copy gee mine -b=20k} allocates a 20K buffer for copying the file \texttt{gee} into the file \texttt{mine}.

For more information about \texttt{copy}, refer to the \textit{Utilities Reference} manual.

Using Procedure Files to Copy Files

The \texttt{dsave} utility copies all files and directories within a specified directory by generating a procedure file. The procedure file is either executed later to actually perform the copy or, by specifying the 
\[ -e \]
on option, executed immediately.

A procedure file is a special OS-9 file containing OS-9 commands. Each command is specified on a line, one command per line. When the procedure file is executed, the OS-9 commands it contains are executed in the order they are listed in the procedure file.

For more information about procedure files, refer to \textit{Chapter 4, The Shell}.

To use the \texttt{dsave} utility, type \texttt{dsave} followed by the pathlist of the directory into which the files are copied, followed by any options you wish to use.
If no pathlist is specified for the destination, the files are copied to the current data directory when the procedure file is executed. If you do not specify the \texttt{-e} option or redirect the output to a file, \texttt{dsave} sends the output to the terminal.

The example below uses the directory structure shown in Figure 2-12.

![Figure 2-12. Dsave Example Directory Structure](image)

If \texttt{PROGMS} is your current data directory and you type \texttt{dsave ../notes}, the following output appears on your screen:

```
$ dsave ../notes
  -t
  chd ../notes
  tmode -w=1 nopause
  load copy
  mkdir MY.PROJ
  chd MY.PROJ
  copy -b=10 /h0/PROGMS/MY.PROJ/progl
  copy -b=10 /h0/PROGMS/MY.PROJ/test.c
  chd ..
  mkdir CONVERSION
  chd CONVERSION
  copy -b=10 /h0/PROGMS/CONVERSION/temp.c
  copy -b=10 /h0/PROGMS/CONVERSION/tally.c
  chd ..
  unlink copy
  tmode -w=1 pause
$
```
Because the output was not redirected to a procedure file and the \texttt{-e} option was not used, the above commands were not executed. They were merely echoed to your screen.

If you now type \texttt{dsave ../notes -e}, the commands are again echoed to the screen. However, the contents of the \texttt{PROGMS} directory are copied into the \texttt{NOTES} directory.

**Selectively Copying Multiple Files**

You can redirect the output of \texttt{dsave} to a file. When you redirect the output, the commands output from \texttt{dsave} are essentially captured in a file. You can later execute this file to actually perform the \texttt{dsave} operation.

To redirect the output from \texttt{dsave} to a file, use the redirection modifier for standard output. The standard output modifier is the greater than (redirect) symbol.

For example, from the \texttt{PROGMS} directory, you can redirect the output from \texttt{dsave} into a file called \texttt{make.bckp} by typing:

\texttt{dsave >make.bckp}

This command creates \texttt{make.bckp} in the current data directory. To perform the \texttt{dsave}, type \texttt{make.bckp} at the command line.

Redirecting the output to a file is helpful when you want to save most, but not all, of the file in the directory or directory being saved. You can edit \texttt{make.bckp} before performing the \texttt{dsave}. This enables you to save only selected files.

Regardless of how you decide to perform the \texttt{dsave}, if \texttt{dsave} encounters a directory file, it automatically creates a new directory and changes to that directory before generating \texttt{copy} commands for files in the subdirectory.

In the \texttt{dsave} example, the directory structure looks like the following diagram when \texttt{dsave} has finished:
If the current working directory is the root directory of the disk, `dsave` creates a file that backs up the entire disk, file by file. This is useful when you need to copy many files from different format disks or from a floppy disk or a hard disk.
Errors During dsave

If an error occurs during the dsave process, the following prompt is displayed:

```
continue (y,n,a,q)?
```

Table 2-8. Responses to dsave Errors.

<table>
<thead>
<tr>
<th>Response</th>
<th>Indicates you…</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>want to continue with dsave</td>
</tr>
<tr>
<td>n</td>
<td>do not want to continue with dsave</td>
</tr>
<tr>
<td>a</td>
<td>want all possible files copied and do not want the prompt displayed on error</td>
</tr>
<tr>
<td>q</td>
<td>want to exit dsave</td>
</tr>
</tbody>
</table>

You can use the -s option to turn off the prompt. This skips any file that cannot be copied and continues the dsave routine without the error prompt.

Indenting for Directory Levels

When you copy several subdirectories, you can use the -i option to indent for directory levels. This helps to keep track of which files are located in which directories.

Keeping Current Directory Backups

You can use dsave to keep current directory backups. Use the -d or -d=<date> options to compare the date of the file to be copied with a file of the same name in the directory where it is to be copied. The -d option copies any file with a more recent date. The -d=<date> option copies any file with a date more recent than that specified. The following example shows the use of dsave with the -d option:

```
$ chd /d0/BACKUP
$ dir
Directory of . 14:14:32
Owner Last Modified Attributes Sector Bytecount Name
------- ------------- ---------- ------ --------- ----
12.4   92/11/12 1417 ---------wr 20CO     11113 program.c
12.4   92/10/05 1601 ---------wr 313D      5744 prog.2
$ chd /d0/WORKFILES
$ dir
```
Chapter 2: The OS-9 File System

Directory of . 14:14:32

<table>
<thead>
<tr>
<th>Owner</th>
<th>Last Modified</th>
<th>Attributes</th>
<th>Sector</th>
<th>Bytecount</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.4</td>
<td>92/11/12</td>
<td>1417</td>
<td>------wr</td>
<td>11113</td>
<td>program.c</td>
</tr>
<tr>
<td>12.4</td>
<td>92/11/12</td>
<td>1601</td>
<td>------wr</td>
<td>5780</td>
<td>prog.2</td>
</tr>
</tbody>
</table>

$ dsave -deb32 /d0/BACKUP
$ chd /d0/BACKUP
$ dir

Directory of . 14:14:32

<table>
<thead>
<tr>
<th>Owner</th>
<th>Last Modified</th>
<th>Attributes</th>
<th>Sector</th>
<th>Bytecount</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.4</td>
<td>92/11/12</td>
<td>1417</td>
<td>------wr</td>
<td>11113</td>
<td>program.c</td>
</tr>
<tr>
<td>12.4</td>
<td>92/11/12</td>
<td>1601</td>
<td>------wr</td>
<td>5780</td>
<td>prog.2</td>
</tr>
</tbody>
</table>

Only prog.2 was copied to the BACKUP directory because the date was more recent in the WORKFILES directory.

For more information about dsave, refer to the Utilities Reference manual.

Deleting Files and Directories

Use the del and deldir utilities to eliminate unwanted file and directories.

- del deletes a file.
- deldir deletes a directory.

If you no longer need a file, deleting the file frees disk space. You must have permission to write to the file or directory in order to delete it.

Deleting Files

To delete a file, type del, followed by the name of the file you want deleted. For example, to delete the file test you created with build, you would type the following command:

```
del test
```

If you execute dir, test is no longer displayed.

When deleting files, you may use wildcards. For example, if you have three files, trial, trial1, and trial.c in a directory, you can use the * wildcard in the command to delete all three files.

Use caution when you use wildcards with utilities like del and deldir. It is easy to unintentionally delete files you want to save.
The `del -p` option displays the following prompt before deleting a file:

```
delete <filename> ? (y,n,a,q)
```

This helps prevent deleting files you want to keep.

**Table 2-9. Responses to Del -p Option**

<table>
<thead>
<tr>
<th>Response</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Delete the file.</td>
</tr>
<tr>
<td>n</td>
<td>Do not delete the file.</td>
</tr>
<tr>
<td>a</td>
<td>Delete specified files without further prompts.</td>
</tr>
<tr>
<td>q</td>
<td>Exit the deleting process.</td>
</tr>
</tbody>
</table>

**Deleting Directories**

To delete a directory, use the `deldir` utility to delete directories. The `deldir` command first deletes all the files and directories in the given directory, and then, if no errors occur, finally deletes the directory name. For example:

```
$ deldir USER2
Deleting directory: USER2
Delete, List, or Quit (d, l, or q) ?
```

**Table 2-10. Responses to Deldir Command**

<table>
<thead>
<tr>
<th>Response</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Delete the directory.</td>
</tr>
<tr>
<td>l</td>
<td>List the directory contents.</td>
</tr>
<tr>
<td>q</td>
<td>Quit without deleting any files.</td>
</tr>
</tbody>
</table>

![Warning]

Do not delete a file or directory unless you are sure you do not need it. Files and directories deleted with the `del` and `deldir` commands are permanently removed.
This chapter describes OS-9 memory modules and module directories. The utilities used with modules and module directories are also discussed. The following sections are included:

- OS-9 Memory Modules
- Module Directories
OS-9 Memory Modules

In addition to organizing your programs and other files into a file system, OS-9 manages both the physical assignment of memory to programs and the logical contents of the memory. To do this, OS-9 uses memory modules.

A memory module is a logical, self-contained program, program segment, or collection of data. Any program or file can become a memory module. Modules are created by compiling and linking programs or by creating data modules. Each module must have three parts:

• A module header contains information that describes the module and its use. The information contained in the module header includes the module name, size, type, language, memory requirements, and entry point.
• A module body contains information such as initialization data, program instructions and constant tables.
• A Cyclic Redundancy Check (CRC) value verifies the module integrity.

In addition to a module header, a module body, and a CRC value, a module must also be position-independent and re-entrant. A re-entrant module does not modify itself; this enables two or more processes to use the module simultaneously. A position-independent module does not depend on being loaded at a specific memory location; this enables OS-9 to load the program wherever memory space is available. In many operating systems, you must specify a load address to place the program in memory. OS-9 determines an appropriate load address only when the program is run.

For more information on modules, refer to the OS-9 Technical Manual.

Using Memory Modules

Memory modules are extremely useful. They provide efficient use of available disk and memory storage, simplify programming jobs, and enable the system to run faster. In addition, they make it easy to customize and adapt OS-9.

An important characteristic of memory modules is that they can be shred by several tasks or users at the same time. For example, if four
users want to run µMACS at the same time, only one copy of the 
µMACS program module is loaded into memory. Other operating 
systems typically load four exact copies of µMACS into memory, 
requiring 300% more memory than if OS-9 were used. In addition, the 
shared module system is completely automatic and usually transparent 
to the user.

Also, with memory modules, frequently used functions can share 
common library modules and you can split large and complex 
programs into smaller, “testable” modules.

**Loading Modules into Memory**

Modules can be loaded into memory during the startup procedure or 
after the system has been accessed. Modules loaded during the startup 
procedure can be loaded either in bootfile or in the startup file. Both of 
these methods for loading modules are discussed in the chapter on 
system management.

Modules necessary for system startup or modules that are used 
frequently should be loaded during the startup procedure. 
Loading modules at system startup places them in contiguous 
spaces of memory; therefore, the memory will be less 
fragmented.

Modules that are used less frequently can be loaded after the system 
startup. To load one or more specified modules into memory, type 
`load`, then the pathlist(s) of the module(s) to be loaded into your 
current module directory. Pathlists may be relative to your current 
execution directory. If the module is located in your current execution 
directory, only the file name is needed after the `load` command:

```
load <file>
```
If `<file>` is not in your execution directory and the shell environment variable `PATH` is defined, `load` searches each directory specified by `PATH` until `<file>` is successfully loaded from a directory. This corresponds to the shell execution search method using the `PATH` environment variable. The names of the modules are added to the module directory. If a module is loaded having the same name as a module already in the current module directory, the module having the highest revision level is kept. The modules are normally loaded from the current execution directory.

Environmental variables are discussed in *Chapter 4, The Shell*.

**Module Security**

The OS-9 file security mechanism enforces certain requirements regarding owner and access permission when loading modules into a module directory. You must have file access permission to the file loaded. If the file is to be loaded from an execution directory, you must have the execute and read permissions for the file. If the file is to be loaded from a directory other than the execution directory, and the `-d` option is used, only the read permission is required.

You must have module access permission to the module to be loaded. This is different from the file access permission of the file containing the target module. The module owner and access permissions are stored in the module header and can be examined by the `ident` utility. To prevent loading super user programs by ordinary users, OS-9 enforces the following restriction: If the module group ID is zero (super group), then the module can be loaded only if the process group ID or the file group ID is zero.

If you are not the owner of a module and not a super user, the public execute and/or public read access permissions must be set. The module access permissions are divided into three groups: the owner, the group, and the public. Only the owner of the module or the super user can set the module access permissions.

There is one other restriction. You must have write permission for the module directory into which you are loading the module. Module directory attributes are discussed later in this chapter.
The Link Count

When modules are loaded into memory, they are added to the module directory structure. Each directory entry contains the module address and a count of the processes using the module. This count is called the link count.

When a process forks to a primary memory module, the module link count is automatically incremented by one. When the process is finished with the primary module, the link count is automatically decremented.

You can also use the link utility to link to a memory module if you want to keep the module in memory. To link to a module, type `link` and the name(s) of the module(s) to be linked. The link count of the specified module is incremented by one each time it is linked.

For example, if you have loaded the module `leap1` into memory, it has a link count of 1. If another user also decides to use `leap1` and links to the memory module, the link count becomes 2.

When you have finished using a module you have linked to with the link utility, remove your link to the module by typing `unlink` and the name(s) of the module(s) to be unlinked. The link count is decremented by 1.

In the example above, if you have finished using `leap1`, type the following command:

```
unlink leap1
```

The link count for `leap1` becomes 1 because another user is still using the module.

The link count becomes 0 if the other user decides to unlink from `leap1`. The module directory entry is deleted and the memory is de-allocated. It is good practice to unlink modules whenever possible to make the most efficient use of available memory resources.

Unless you have explicitly linked to a module using `link`, you do not need to unlink the module.
**Modules Remaining in Memory**

There are three cases when a module is not removed from memory, even if the module’s link count reaches 0:

- Modules have been loaded during system bootstrap.
- The modules being loaded are sticky modules.
- Modules are still in use.

Modules loaded during system bootstrap cannot be unlinked from memory regardless of their link count. It is potentially fatal to your system to unlink memory modules such as the kernel.

A sticky module sticks in the system even when it has a link count of 0. A sticky module is removed from memory only when `unlink` is used to lower the module link count to -1. You can use the `fixmod` utility to make a module sticky. Generally, sticky modules are modules used frequently enough to warrant them staying in the system at all times.

The third case involves modules with their link counts lowered to 0 (or -1 for sticky modules), but are still in use. For example, if one user is using µMACS and another user lowers µMACS’ link count to 0, the module stays in memory because the module is still in use.

**Module Directories**

OS-9 is unique because memory modules may be arranged in a hierarchical directory structure just like files and directories. Therefore, when you load a module into memory, you must make a decision as to which module directory should contain the module.

Immediately after OS-9 is booted, a single module directory is created in which all of the modules were loaded during system startup—unless either `sysgo` or the `startup` file has been modified to build a memory module directory structure. You may create additional module directories at any time. This enables you to organize modules in memory. Each module directory can contain other module directories.
OS-9 enables you to load modules into specific directories, even if a module of the same name is loaded into another directory. This means you can make changes to a program and load it into your own module directory. Once in the module directory, the module can be accessed instead of a module with the same name elsewhere in the module directory system. From this directory, you can test and debug the module without affecting other system users.

For example, if you are using a module called mine that is loaded into your module directory, another user could be using or developing another mine module in a different directory.

Module directories also enable you to load programs into memory without the programs becoming known to the public.

Current Module Directory

Memory module directories are similar to other directories, as you can specify a current module directory. The current module directory is important for accessing memory modules.

For example, when modules are loaded into memory, they are added to the process current module directory. Likewise, when a process forks a new process, OS-9 searches the current module directory for the target module first. If the search fails, OS-9 searches the process' alternate module directories. Failing to find the module in memory, OS-9 attempts to load the target module into the current module directory.
You can set the initial current module directory in your `.login` file. Use the `MDPATH` environment variable in the `.login` file to establish the alternate module directory. You can change the current memory module directory using the `chm` built-in shell command. To change module directory, type `chm` and the pathlist to the new module directory.

You can use full or relative pathlists when specifying module directory pathlists. However, pathlists beginning from the root module directory begin with a single slash (`/`). Pathlists beginning with either two slashes (`//`) or no slash specify the pathlist begins at the current module directory.

For example, the following pathlist begins at the root module directory:

```
chm /user/paul
```

The next two commands both begin at the your current module directory:

```
chm //doc/proj1
chm doc/proj1
```

If the `MDHOME` environment variable is set, typing `chm` with no pathlist changes your current module directory to the directory specified by the `MDHOME` environment variable.

For more information on the `chm` built-in shell command, refer to the Utilities Reference manual.

### Displaying the Contents of Module Directories

You can display the contents of memory module directories with the `mdir` utility. To see the contents of a particular memory module directory, type `mdir` and the pathlist to the module you want to display. Pathlists may be either full or relative.

For example, to display the contents of the `UTILS` module directory located in the root module directory, type the following command:

```
mdir /utils
```

A screen containing the contents of the `UTILS` folder should display:

```
Module Directory of /utils
<FILE1>       <FILE2>       <FILE3>        csl
dir
```
To display an extended listing of a module directory, use the \texttt{-e} option. The extended listing displays detailed information concerning each module located in the directory. The following is an example of a \texttt{mdir -e} command.

\begin{verbatim}
mdir //doc -e
Module Directory of //doc


Addr   Size  Owner   Perm  Type Revs  Ed  #  Lnk  Module name
------ -------- ----------- ---- ---- ---- ----- ----- ------------------
36a170  1940  22.148   0333 MDir 0000  0  1 <FILE1>
2f90f0  7948   7.17    0555 MDir a000  7  2 <FILE2>
2adda0  1834   0.22    0555 MDir 8001  7  1 RIC
033a68  45408  22.148   0555 Subr c000  18  7 csl
318f20  23402  1.169   0555 Prog c001  36  0 dir
\end{verbatim}

\textbf{Memory Module Directory Attributes}

You can examine and change module attributes using the \texttt{mdattr} utility. To use the \texttt{mdattr} utility, type \texttt{mdattr} and the module directory pathlist. For example:

\begin{verbatim}
mdattr leap1
---r---r--wr  leap1
\end{verbatim}

Memory module directories can have owner, group, and public attributes. These attributes are each divided into four fields (from right to left):

- read attribute
- write attribute
- reserved
- reserved

The attribute abbreviations are listed in \textbf{Table 3-1}.

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
\textbf{Abbreviation} & \textbf{Means} \\
\hline
\texttt{r} & owner read permission \\
\texttt{w} & owner write permission \\
\texttt{gr} & group read permission \\
\texttt{gw} & group write permission \\
\texttt{pr} & public read permission \\
\texttt{pw} & public write permission \\
\hline
\end{tabular}
\caption{Attribute Abbreviations}
\end{table}
A module directory with all permissions set looks like similar to that shown below:

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

The first `wr` series is the public read and write permissions. The second is the group read and write permission. The third is the owner read and write permissions. The hyphens (-) are place holders for reserved fields.

A permission is changed by giving its abbreviation preceded by a hyphen (-). It is turned off by preceding its abbreviation with a hyphen followed by the letter n (-n). Permissions not explicitly named are not affected. If no permissions are specified, the current file attributes are printed.

To see the attributes of the module `leap1`, type the following command:

```
$ mdattr leap1
------wr--wr  leap1
```

`leap1` has the group and owner read and write permissions set. To remove the group write permission and add the public read permission to `leap1`, type the following command:

```
$ mdattr leap1 -ngw -pr
---r---r--wr  leap1
```

**Creating New Memory Module Directories**

To create new memory module directories, use the `makmdir` utility. The `makmdir` utility creates the new module directory in the directory specified. To create a new memory module directory, type `makmdir` followed by the module directory pathlist specifying the new module directory.
The following example uses this memory module directory structure:

**Figure 3-2. Before makmdir Command**

![Diagram](image)

To create the directory `TONY` in the `USER` directory, type the following command:

`makmdir /user/TONY`

The module directory structure looks like the following figure:

**Figure 3-3. After makmdir Command**

![Diagram](image)

`makmdir` creates the new module directory with the read and write permissions set for the owner, group, and public. In addition, `makmdir` only searches the current module directory for a specified module path when creating a new module directory. The alternate pathlists specified by the `MDPATH` environment variable are not searched if a specified module path is not found in the current module directory.

For example, if `USER` is your current module directory and you want to make a new directory in a directory called `TEST`, OS-9 does not search the alternate module directories for a module directory named `TEST`. 
Deleting Memory Module Directories

You can delete memory module directories using the `delmdir` utility. To delete a module directory, type `delmdir`, the pathlist for the module directory, and any desired options.

If the module directory to be deleted contains sub-directories, the sub-directories are also deleted. For example, if the `USER` directory in the previous example is deleted, the directories `AMY`, `TONY`, and `JESSICA` are also deleted.

delmdir searches only the current module directory for a specified module path when deleting a module directory. The alternate pathlists specified by the `MDPATH` environment variable are not searched if a specified module path is not found in the current module directory.

Modules within the directory to be deleted or any of its sub-directories must not be in use. If a module in a directory is in use when `delmdir` is called, `delmdir` is not successful. You must also have the appropriate access permissions to a module directory in order to delete it.
This chapter contains a detailed description of the shell, the OS-9 user interface. It includes the following sections:

- The Function of the Shell
- The Shell Environment
- Built-In Shell Commands
- Shell Command Line Processing
- Shell Procedure Files
- Time-Sharing System Startup Procedure File
- Creating a Temporary Procedure File
- Multiple Shells
- Waiting for Background Procedures
- Command History
- Error Reporting
The Function of the Shell

The shell is the OS-9 command interpreter program. The shell takes the commands you enter and translates them into commands the operating system understands and executes.

The shell also provides an environment that can be configured by the user; this allows you to personalize the way OS-9 works on your system. You can use the shell to change the shell prompt, send error messages to a file, or backup your disk before you log out.

The shell command starts the shell program. This command is automatically executed following system startup or after logging on to a timesharing terminal. When the shell is ready for commands, it displays the following prompt:

```
[1]$ 
```

The [1] in the prompt is the history number for that command line. This has been omitted from the rest of the prompts shown in this manual. The command line history is discussed in this chapter.

This prompt indicates the shell is active and waiting for a command from your keyboard. From here you can type a command line followed by a carriage return.

Shell Options

A number of options are available to the shell. By default, some are automatically turned on following startup or log on. The available shell options are listed in Table 4-1.

Table 4-1. Shell Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>Echo the command line if it is altered after if is entered.</td>
</tr>
<tr>
<td></td>
<td>This is the default option.</td>
</tr>
<tr>
<td>-c=&lt;num&gt;</td>
<td>Specify the number of previously executed commands the shell should remember.</td>
</tr>
<tr>
<td></td>
<td>This provides a history of your commands. If &lt;num&gt; is not specified, the default is 40.</td>
</tr>
</tbody>
</table>
### Table 4-1. Shell Options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-e=&lt;file&gt;</code></td>
<td>Print error messages from <code>&lt;file&gt;</code>. If no file is specified, <code>/dd/SYS/errmsg</code> is used. Without this option, the shell prints only error numbers with a brief message description. Each error is described in the appendix on error codes in the OS-9 Technical Manual.</td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Display the command’s history number in front of the command line prompt. This is the default option.</td>
</tr>
<tr>
<td><code>-l</code></td>
<td>The <code>logout</code> built-in command is required to terminate the login shell. <code>&lt;eof&gt;</code> does not cause the shell to terminate.</td>
</tr>
<tr>
<td><code>-na</code></td>
<td>Does not echo the command line if it is altered after it is entered.</td>
</tr>
<tr>
<td><code>-nc</code></td>
<td>Does not keep track of your command history.</td>
</tr>
<tr>
<td><code>-ne</code></td>
<td>Print no error messages. This is the default option.</td>
</tr>
<tr>
<td><code>-nh</code></td>
<td>Does not display the command’s history number.</td>
</tr>
<tr>
<td><code>-nl</code></td>
<td><code>&lt;eof&gt;</code> terminates the login shell. <code>&lt;eof&gt;</code> is normally caused by pressing the <code>&lt;Esc&gt;</code> key. This is the default option.</td>
</tr>
<tr>
<td><code>-np</code></td>
<td>Does not display the prompt.</td>
</tr>
<tr>
<td><code>-nq</code></td>
<td>Does not keep assigns in environment.</td>
</tr>
<tr>
<td><code>-ns</code></td>
<td>Does not save your command history from one login session to the next. This is the default option.</td>
</tr>
<tr>
<td><code>-nt</code></td>
<td>Does not echo input lines. This is the default option.</td>
</tr>
<tr>
<td><code>-nv</code></td>
<td>Turns off verbose mode. This is the default option.</td>
</tr>
<tr>
<td><code>-nx</code></td>
<td>Does not abort process on error.</td>
</tr>
</tbody>
</table>
You can change shell options with either of two methods. The two methods accomplish the same function.

1. Type the option on the command line or after the `shell` command. For example:
   - `$ -np` turns off the shell prompt.
   - `$ shell -np` creates a new shell that does not prompt. When the new shell is exited, the original shell prompts.

2. Use `set`, a special shell command. To set shell options, type `set`, followed by the options desired. When using the `set` command, a hyphen (-) is unnecessary before the letter option. For example:
   - `$ set np` turns off the shell prompt.
   - `$ shell set p` creates a new shell that does not prompt. When the new shell is exited, the original shell prompts.

### Table 4-1. Shell Options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p</td>
<td>Displays the prompt. The default prompt is a dollar sign ($).</td>
</tr>
<tr>
<td>-p=&lt;string&gt;</td>
<td>Set the current shell prompt equal to <code>&lt;string&gt;</code>.</td>
</tr>
<tr>
<td>-q</td>
<td>Keep assigns in environment.</td>
</tr>
<tr>
<td>-s</td>
<td>Save your command history from one login session to the next.</td>
</tr>
<tr>
<td></td>
<td>The command history is saved in a <code>.history</code> file in your home directory.</td>
</tr>
<tr>
<td>-t</td>
<td>Echo input lines.</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose mode: display a message for each directory searched when executing a command.</td>
</tr>
<tr>
<td>-x</td>
<td>Abort process on error.</td>
</tr>
</tbody>
</table>

This is the default option.
The Shell Environment

The shell maintains a unique list of environment variables for each user on an OS-9 system. These variables affect the operation of the shell or other programs subsequently executed and can be set according to your preference.

All environment variables can be accessed by any process called by the shell or descendant shells. This enables you to use the environment variables as global variables.

If an environment variable is redefined by a subsequent shell, the variable is only redefined for that shell and its descendents. The environment variable is not redefined for the parent shell.

The environment variables shown in Table 4-2 are automatically set up when you log on to a time-sharing system.

Table 4-2. Environment Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT</td>
<td>The name of the terminal. An example of a valid name is /t1. The tsmon utility automatically sets up PORT.</td>
</tr>
<tr>
<td>HOME</td>
<td>Your home directory. The home directory is specified in your password file entry and is your current data directory when you first log on to the system. This is also the directory used when the command chd with no parameter is executed.</td>
</tr>
<tr>
<td>SHELL</td>
<td>The first process executed upon logging on to the system.</td>
</tr>
<tr>
<td>USER</td>
<td>The user name you type when prompted by the login command.</td>
</tr>
<tr>
<td>PATH</td>
<td>Specify any number of directories. Directory paths must be separated by a colon (:). The shell uses PATH as a list of command directory to search when executing a command. If the default commands directory does not include the file or module to be executed, each directory specified by PATH is searched until the file/module is found or the list is exhausted.</td>
</tr>
</tbody>
</table>
For single user systems, these variables can be set with the `setenv` command. A procedure file may also be set up with your normal configuration of these variables. This procedure file could then be executed each time you start up your terminal.

Other environment variables include those shown in Table 4-3:

**Table 4-3. Optional Environment Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MDHOME</strong></td>
<td>Specify your home module directory. This is the module directory used when executing the command <code>chm</code> with no parameter.</td>
</tr>
<tr>
<td><strong>MDPATH</strong></td>
<td>Specify any number of module directories to search. Module directory paths must be separated by a colon (<code>:</code>). The shell uses <code>MDPATH</code> as a list of module directories to search when executing a command.</td>
</tr>
<tr>
<td><strong>PROMPT</strong></td>
<td>Specify the current prompt. By specifying an at sign (<code>@</code>) as the first character of your prompt, you may easily keep track of how many shells you have running under each other. <code>@</code> is used as a replaceable macro for the shell level number. The base level is set by the environment variable <code>_sh</code>.</td>
</tr>
<tr>
<td><code>_sh</code></td>
<td>Specify the base level for counting the number of shell levels. For example, set the shell prompt to <code>@howdy:</code> and <code>_sh</code> to 0:</td>
</tr>
</tbody>
</table>

```
$ setenv _sh 0
$ -p="@howdy: "
howdy: shell
1.howdy: shell
2.howdy: eof
1.howdy: eof
howdy:
```

| **TERM** | Specify the type of terminal being used. **TERM** allows word processors, screen editors, and other screen dependent programs to know what type of terminal configuration is used. |

Environment variables are case sensitive. OS-9 cannot recognize a variable if the proper case is not used.
Changing the Shell Environment

Three commands are available for use with environment variables: `setenv`, `unsetenv`, and `printenv`. These variables are only known to the shell in which they are defined and any descendant processes from that shell.

Table 4-4. Environment Variable Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setenv</code></td>
<td>Declare the variable and sets its value. The variable is put in an environment storage area accessed by the shell. For example:</td>
</tr>
<tr>
<td></td>
<td>$ setenv PATH ..:/h0/cmds:/d0/cmds:/dd/cmds</td>
</tr>
<tr>
<td></td>
<td>$ setenv _sh 0</td>
</tr>
<tr>
<td></td>
<td><code>setenv</code> does not change the environment of the parent process of the shell from which <code>setenv</code> was issued.</td>
</tr>
<tr>
<td><code>unsetenv</code></td>
<td>Clear the value of the variable and removes it from storage. For example:</td>
</tr>
<tr>
<td></td>
<td>$ unsetenv PATH</td>
</tr>
<tr>
<td></td>
<td>$ unsetenv _sh</td>
</tr>
<tr>
<td><code>printenv</code></td>
<td>Print the variables and their values to standard output. For example:</td>
</tr>
<tr>
<td></td>
<td>$ printenv</td>
</tr>
<tr>
<td></td>
<td>PATH ..:/h0/cmds:/d0/cmds:/dd/cmds</td>
</tr>
<tr>
<td></td>
<td>PROMPT howdy</td>
</tr>
<tr>
<td></td>
<td>_sh 0</td>
</tr>
</tbody>
</table>

These three commands are described in the Utilities Reference manual.

Using Environmental Variables as Command Line Parameters

When you use the following syntax, the shell replaces the environment variable with its value:

```
$(<env var>)
```

For example, if HOME is set to `/h0/USR/ROB` and you enter the command `dir $(HOME)`, the shell executes the command `dir /h0/USR/ROB`. 
This substitution is useful for entire command lines. By using `setenv`, a command line can be assigned to an environment variable:

```bash
setenv PR "procs -ea"
```

Any time `$(PR)` appears on the command line, the shell automatically substitutes `procs -ea`.

## Built-In Shell Commands

The shell has a special set of commands or option switches built in to the shell. These commands are executed without loading a program and creating a new process. They can be executed regardless of your current execution directory.

The built-in commands and their functions are as listed in Table 4-5.

### Table 4-5. Built-in Shell Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>* &lt;text&gt;</code></td>
<td>Indicates a comment: <code>&lt;text&gt;</code> is not processed.</td>
</tr>
<tr>
<td><code>assign</code></td>
<td>Allows you to assign commands and strings to a single word for command line substitutions.</td>
</tr>
<tr>
<td><code>chd &lt;path&gt;</code></td>
<td>Change the current data directory to the directory specified by <code>&lt;path&gt;</code>.</td>
</tr>
<tr>
<td><code>chm &lt;path&gt;</code></td>
<td>Change the current module directory to the module directory specified by <code>&lt;path&gt;</code>.</td>
</tr>
<tr>
<td><code>chx &lt;path&gt;</code></td>
<td>Change the current execution directory to the directory specified by <code>&lt;path&gt;</code>.</td>
</tr>
<tr>
<td><code>ex &lt;name&gt;</code></td>
<td>Directly execute the named program. This replaces the shell process with a new execution module.</td>
</tr>
<tr>
<td><code>hist</code></td>
<td>Display the history of your commands.</td>
</tr>
<tr>
<td><code>kill &lt;proc ID&gt;</code></td>
<td>Abort the process specified by <code>&lt;proc ID&gt;</code>.</td>
</tr>
<tr>
<td><code>logout</code></td>
<td>Terminate the current shell.</td>
</tr>
</tbody>
</table>

If the login shell is to be terminated, the `.logout` file in the home directory is executed and then the login shell is terminated.
Chapter 4: The Shell

Shell Command Line Processing

The shell reads and processes command lines one at a time from its input path (usually your keyboard). Each line is first parsed to identify and process any of the following parts that may be present:

Table 4-6. Command Line Parts

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>name of a program, procedure file, built-in command, or pathlist</td>
</tr>
<tr>
<td>parameters</td>
<td>names of files, programs, values, variables, and constants to be passed to the program being executed</td>
</tr>
<tr>
<td>execution modifiers</td>
<td>These modify a program’s execution by redirecting I/O or changing the priority or memory allocation of a process.</td>
</tr>
<tr>
<td>separators</td>
<td>When multiple commands are placed on the same command line, separators specify whether they should be executed sequentially or concurrently.</td>
</tr>
</tbody>
</table>

Only the keyword needs to be present for the shell to process a command line. Parameters, execution modifier, and separators are optional. After the keyword has been identified, the shell processes any execution modifiers and separators. Any text not yet processed is assumed to be parameter and is passed to the program called.
The keyword must be the first word in the command line. If the keyword is a built-in command, it is executed immediately. If the keyword is not a built-in command, the shell assumes it is a program name and attempts to locate it. The shell searches for the command in the following sequence:

1. The shell checks the memory to see if the program has already been loaded into the module directory. If it is already in memory, there is no need to load another copy. The shell then calls the program to be executed.

2. If the program is not in memory, your current execution directory is searched. An attempt to load the program is made if it is found. If this fails, the shell tries to execute it as a procedure file. If this fails, the shell attempts the same procedure using the next directory specified in the PATH environment variable. This continues until the command is successfully executed or the list of directory is exhausted.

3. Your current data directory is searched. If the specified file is found, it is processed as a procedure file. Procedure files are assumed to contain one or more shell command lines. These command lines are processed by a newly created, or child shell as if they had been typed in manually. After all commands from the procedure files are executed, control returns to the old, or parent shell. Because the commands are processed by the child shell, all built-in commands in the procedure file such as chd and chx only affect the child shell.

An error is returned if the program is not found. If the program is found and executed, the shell waits until the program terminates. When the program terminates, it reports any errors returned. If there are more input lines, the shell gets the next line and the process is repeated.

This sample command line calls a program:

```plaintext
$ prog #12K sourcefile -l -j >p
```

In this example:

- `prog` is the keyword.
- `#12K` is a modifier requesting an alternate memory size be assigned to this process. In this case, 12K is used as memory.
- `sourcefile -l -j` are parameters passed to `prog`. 
> is a modifier redirecting output to a file or device.

In this case, > redirects the output to the printer (/p).

/p is the system printer.

**Special Command Line Features**

In addition to basic command line processing, the shell facilitates the following jobs:

- memory allocation
- I/O redirection, including filter
- process priority
- wildcard pattern matching
- multi-tasking; concurrent execution

These functions are accessed by using execution modifiers, separators, and wildcard characters. The combination of ways you can use these capabilities is virtually unlimited.

Characters comprising execution modifiers, separators, and wildcards are stripped from the part(s) of the command line passed to a program as parameter. Characters cannot be passed as parameters to programs unless contained in quotes.

**Table 4-7. Execution Modifiers**

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Additional memory size</td>
</tr>
<tr>
<td>^</td>
<td>Process priority</td>
</tr>
<tr>
<td>&gt;</td>
<td>Redirect output</td>
</tr>
<tr>
<td>&lt;</td>
<td>Redirect input</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Redirect error output</td>
</tr>
</tbody>
</table>

**Table 4-8. Separators**

<table>
<thead>
<tr>
<th>Separator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>;</td>
<td>Sequential execution</td>
</tr>
<tr>
<td>&amp;</td>
<td>Concurrent execution</td>
</tr>
</tbody>
</table>
Execution Modifiers

The shell processes execution modifiers before the program is run. If an error is detected in any of the modifiers, the run is aborted and the error reported.

Additional Memory Size Modifier

Every executable program is converted to machine language for storage. During the conversion process, a module header is created for the program. A module header is part of all executable programs and holds the program’s name, size, memory requirements, and other details.

A complete explanation of module headers is available in the OS-9 Technical Manual.

When an executable program is processed by the shell, the minimum amount of working memory specified in the program module header is allocated. To increase the default memory size, memory can be assigned in 1K increments using the pound sign modifier (#) followed by a number of allocated kilobytes: #10k or #10. The shell adds the allocated number of kilobytes to the default listed in the program header.

<table>
<thead>
<tr>
<th>Table 4-8. Separators (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Separator</strong></td>
</tr>
<tr>
<td>+</td>
</tr>
<tr>
<td>!</td>
</tr>
<tr>
<td>!!</td>
</tr>
<tr>
<td>!!!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4-9. Wildcards</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wildcard</strong></td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>?</td>
</tr>
</tbody>
</table>
The increase in memory allocation only affects one command. If you want to increase the allocation for the next command, you must add the modifier (\#) again.

\[\text{Programs written in C use extra memory for stack space only.}\]

**I/O Redirection Modifiers**

Redirection modifiers redirect the program’s standard I/O paths to alternate file or devices. Usually, programs do not use specific file or device names. This makes the redirection of standard I/O to any file or device fairly simple without altering the program. Programs normally receiving input from a terminal or sending output to a terminal use one or more of these standard I/O paths:

- **Standard Input Path**: Pass data from a keyboard to a program.
- **Standard Output Path**: Pass output data from a program to a display.
- **Standard Error Path**: This can be used for either input or output, depending on the nature of the program using it. This path is commonly used to output routine status messages, such as prompts and errors to the terminal’s display. By default, the standard error path uses the same device as the standard output path.

A new process can only be created by an existing process. The new process is known as the “child process”. The process creating the child process is known as the “parent process”. Each child process inherits the standard I/O paths from the parent process.

When the shell creates a new process, it inherits the shell’s standard I/O paths. Upon startup or login, standard input is the terminal keyboard. The standard output and standard error are directed to the display. Consequently, the child process standard input is the keyboard. The child process standard output and standard error are directed to the display.
Below are the three redirection modifiers:

- `<` Redirect the standard input path.
- `>` Redirect the standard output path.
- `>>` Redirect the standard error path.

When you use a redirection modifier on a shell command line, the shell opens the corresponding paths and passes them to the new process as its standard I/O paths. When you use redirection modifiers on a command line, they must be immediately followed by a path describing the file or device to or from which the I/O is to be redirected.

**Standard Devices**

Each physical input/output device supported by the system must have a unique name within a module directory. Although the device names used on a system are somewhat arbitrary, it has become customary to use the names Microware assigns to standard devices in OS-9 packages. The standard devices are shown in Table 4-10.

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>term</code></td>
<td>Primary system terminal</td>
</tr>
<tr>
<td><code>t1, t2, etc.</code></td>
<td>Other serial terminals</td>
</tr>
<tr>
<td><code>p</code></td>
<td>Parallel printer</td>
</tr>
<tr>
<td><code>p1</code></td>
<td>Serial printer</td>
</tr>
<tr>
<td><code>dd</code></td>
<td>Default disk drive</td>
</tr>
<tr>
<td><code>d0</code></td>
<td>Floppy disk drive unit 0</td>
</tr>
<tr>
<td><code>d1, d2, etc.</code></td>
<td>Other floppy disk drives</td>
</tr>
<tr>
<td><code>h0, h1, etc.</code></td>
<td>Hard disk drives (format-inhibited)</td>
</tr>
<tr>
<td><code>h0fmt, h1fmt, etc.</code></td>
<td>Hard disk drives (format-enabled)</td>
</tr>
<tr>
<td><code>n0, n1, etc.</code></td>
<td>Network devices</td>
</tr>
<tr>
<td><code>mt0, mt1</code></td>
<td>Tape devices</td>
</tr>
<tr>
<td><code>r0</code></td>
<td>RAM disk</td>
</tr>
</tbody>
</table>

The `h0fmt` and `h1fmt` device descriptors have a bit set allowing you to use the `format` and `os9gen` utilities on them. To avoid accidentally formatting a hard disk, you should use the device names `h0, h1, etc.`
Device names may only be used as the first name of a pathlist and must be preceded by a slash (/) to indicate the name is an I/O device. If the device is not a mass storage multi-file device like a disk drive, the device name must be the only name in the path. This restriction is true for devices such as terminals and printers.

For example, the standard output of `list` can be redirected to write to the system printer instead of the terminal:

```
$ list correspondence >/p
```

Files referenced by I/O redirection modifier are automatically opened or created and closed as appropriate by the shell. In the below example, the output of `dir` is redirected to the path `/d1/savelisting`:

```
$ dir >/d1/savelisting
```

If `list` is used on the path `/d1/savelisting`, output from `dir` is displayed as follows:

```
$ List /d1/savelisting
   directory of .  10:15:00
   file1           myfile          savelisting
```

You can use redirection modifiers before and/or after the program parameter, but you can use each modifier only once in a given command line. Redirection modifiers can be used together to cause more than one of the standard paths to be redirected. For example, `shell <>>>/t1` redirects all three standard paths to `/t1`.

The plus and hyphen characters (+ and -) can be used with output style redirection modifier. The `>-` modifier redirects output to a file. If the file already exists, the output overwrites it. The `>+` modifier adds the output to the end of the file. The following example overwrites `dirfile` with output from the execution directory listing:

```
dir -x >-dirfile
```

The below example adds the listing of `newfile` to the end of `oldfile`.

```
list newfile >+oldfile
```

Spaces must not occur between redirection operators and the device or file path.
Process Priority Modifier

On multi-user systems or when multi-tasking, many processes seem to be simultaneously executed. Actually, OS-9 uses a scheduling algorithm to allocate execution time to active processes.

All active processes are sorted into a queue based on the age of the process. The age is a number between 0 and 65535 based on how long a process has waited for execution and its initial priority.

On a timesharing system, the system manager assigns the initial priority for processes started by each user. This priority for the initial process is listed in the password file. The initial process is usually the shell. On a single user system, processes have their priority set in the Init module. All child processes inherit the parent process priority.

When a process enters the active queue, it has an age set to its initial priority. Every time a new active process is submitted for execution, all earlier processes’ ages are incremented. The process with the highest age is executed first.

If you want a program to run at a higher priority, use the caret modifier (^). By specifying a higher priority, a process is placed higher in the execution queue. For example:

```
$ format /d1 ^255
```

In this example, the process format is assigned a priority of 255. By assigning a lower number, a lower priority can be specified.

Specifying too high of a priority for a process can cause all other processes to be locked out until their ages mature.

For example, if you specify a priority of 2000 for a program and all the other processes have an age of less than 100, your program is the only process executed on the system until either your program terminates or another process’ age reaches 2000. If another process’ age reaches 2000, it runs once and enters back in the queue at its initial priority. Once again, your program either runs until it terminates or until another process’ age reaches 2000.
Wildcard Matching

The shell uses some alternate ways to identify file and directory names. The shell accepts wildcards in the command line. The two recognized wildcard characters are the asterisk (\*) and the question mark (?).

An asterisk (*) matches any group of zero or more characters. A question mark (?) matches any single character. The shell searches the current data directory or the directory given in a path for matching file names.

For the following examples, a directory containing the following file is used:

directory of FILES  14:45:20
diary  diary2  form  form.backup  forms
login.names  logistics  logs  old  oldstuff
setime.c  shellfacts  sizes  sizes.backup  utils1

The command list log* lists the contents of login.names, logistics, and logs. The pattern log* matches all file names beginning with log followed by zero or more characters. The following commands demonstrate the function of this wildcard.

Table 4-11. Commands Using * Wildcards

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>list s*</td>
<td>List all files in the current data directory beginning with s: shellfacts, setime.c, and sizes.</td>
</tr>
<tr>
<td>del *</td>
<td>Delete every file in the current data directory (in this example, FILES).</td>
</tr>
<tr>
<td>dir ../*.backup</td>
<td>List all files in the parent directory ending with .backup.</td>
</tr>
<tr>
<td>dir -x d*</td>
<td>List all files in the current execution directory starting with the letter d. This can be helpful if you are unsure of the spelling of a particular utility.</td>
</tr>
</tbody>
</table>
The question mark (\?) matches any single character in the position where the wildcard character is located. For example, the command line `list log?` only lists the contents of the file `logs`. The following commands demonstrate the function of this wildcard.

**Table 4-12. Commands Using ? Wildcards**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>del form?</code></td>
<td>Delete the file <code>forms</code> but not <code>form</code>.</td>
</tr>
<tr>
<td><code>list s????</code></td>
<td>List the contents of <code>sizes</code>, but not <code>setime.c</code> or <code>shellfacts</code>.</td>
</tr>
</tbody>
</table>

In both examples, the shell searches only for names with five characters.

Wildcards may also be used together. For example, the command `list *.?` lists any files ending in a period followed by any letter, number or special character, regardless of what comes before the period. In this case, `list *.?` lists the contents of the file `setime.c`.

The shell only attempts to expand a character string containing a wildcard if the character string could be a pathlist. The shell does not expand wildcards used in the keyword of a command line. For example, the shell does not expand the asterisk in the following:

```
d* forms
```

The shell disregards wildcard characters enclosed in double quotes. For example:

```
echo "*"
```

This echoes an asterisk (\*) to standard output (usually the terminal). If the double quotes around the asterisk were left out, the shell expands the wildcard to include every file name in the current directory and outputs each name to the terminal.

⚠️ You must be careful when using wildcards with utilities such as `del` and `deldir`. Wildcards should not be used with the `-x` or `-z` options of most utilities.
Command Separators

A single shell input line can include more than one command line. These command lines may be executed sequentially or concurrently. Sequential execution causes one program to complete its function and terminate before the next program is allowed to begin execution. Concurrent execution allows several command lines to begin execution and run simultaneously.

Commands can be sequentially executed by separating the command with a semicolon (;). Commands can be concurrently executed by separating the commands with an ampersand (&) or plus sign (+).

Sequential Execution

When one command per line is entered from the keyboard, programs are executed one after another, or sequentially. All programs executed sequentially are individual processes created by the shell. After initiating a sequentially executed program, the shell waits until the program it created terminates. The command line prompt does not return until the program has finished.

For example, the following command lines are executed one after another. The copy command is executed first, followed by the dir command.

```bash
$ copy myfile /D1/newfile
$ dir >/p
```

You can specify more than one program on a single shell command line for sequential execution by separating each program name and its parameter from the next one with a semicolon (;). For example:

```bash
$ copy myfile /D1/newfile; dir >/p
```

The shell first executes copy and then dir. The command line executes exactly as the previous two command lines unless an error occurs.

If an error is returned by any program, subsequent commands on the same line are not executed regardless of the \(-nx\) option. In all other regards, a semicolon (;) and a carriage return act as identical separators.
The following example copies the contents of oldfile into newfile. When the copy command finished, oldfile is deleted. Then the contents of newfile are listed.

$ copy oldfile newfile; del oldfile; list newfile

In the next example, the output from dir is redirected into myfile in the d1 directory. The output from list is then redirected to the printer. Finally, temp is deleted.

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

Multi-tasking: Concurrent Execution

Programs may be executed concurrently using the ampersand (&) or plus sign (+) separators. This allows programs to run at the same time as other programs, including the shell. The shell does not wait to complete a process before processing the next command. Concurrent execution is how a background program is started.

Multi-tasking is accomplished by using the concurrent execution separators. The number of programs that can run at the same time is not fixed; it depends upon the amount of free memory in the system and the memory requirements of the specific programs.

Below is an example:

$ dir >/P& list file1& copy file1 file2; del temp

The dir, list, and copy utilities run concurrently because they were separated by an ampersand (&). del does not run until copy has terminated because sequential execution (;) was specified.

By adding an ampersand (&) or plus sign (+) to the end of a command line, regardless of the type of execution specified, the shell immediately returns command to the keyboard, displays the $ prompt, and waits for a new command. This frees you from waiting for a process or sequence of processes to terminate.

This is especially useful when making a listing of a long text file on a printer. Instead of waiting for the listing to print to completion, using either of the concurrent execution separators allows you to use your time more efficiently.
The plus sign (+) separator allows you to fork a process to run in the background as an orphan process. An orphan process does not have a parent process. This means regardless of how the process terminates, you are not notified. Also, when the `wait` command is executed, the shell does not wait for the process to finish execution. Executing an orphan process is useful for executing non-terminating processes.

For example, you could execute `tsmon` and any networking utilities concurrently using the plus sign separator:

```
$ tsmon /t1 +
```

`tsmon` is started, but your shell is not considered to be the parent process.

If you have several processes running at once, you can display a status summary of all your processes with the `procs` utility. `procs` gives you a complete list of your current processes and pertinent information about each process.

```
procs
```

The `procs` utility is discussed later in this chapter in the section The `procs` Utility.

### Pipes and Filters

The third kind of separator is the exclamation point (!) used to construct "pipelines". Pipelines consist of two or more concurrent programs whose standard input and/or output paths connect to each other using "pipes". A pipe is simply a way to connect the output of a process to the input of another process, so the two run as a sequence of process: a pipeline. Pipes are one of the primary means for transferring data from process to process for inter process communications. Pipes are first-in, first-out buffers.

All programs in a pipeline are executed concurrently. The pipes automatically synchronize the programs so the output of one never gets ahead of the input request of the next program in the pipeline. This ensures data cannot flow through a pipeline any faster than the slowest program can process it.

Any program that reads data from standard input can read from a pipe. Any program that writes data to standard output can write data to a pipe. Several utilities are designed so the standard output of one can be piped to the standard input of another.
For example:

$\texttt{dir \ -e \ !\ \ pr}$

This example causes the standard output of \texttt{dir} to be piped to the standard input of the \texttt{pr} utility instead of on the terminal screen. \texttt{pr} reads the output of \texttt{dir} even though \texttt{pr} reads standard input by default. \texttt{pr} then displays the result.

In \textbf{Figure 4-1} the standard output of the \texttt{dir \ -e} command is piped to the standard input of the \texttt{pr} command through an un-named pipe. The \texttt{pr} utility displays the results of the \texttt{dir \ -e} command.

\textbf{Figure 4-1. Unnamed Pipe}

![Figure 4-1. Unnamed Pipe](image)

The \texttt{pr} command may be modified with the following options:

- Two exclamation points (\texttt{!!}) pipe the standard error from one program to another.
- Three exclamation points (\texttt{!!!}) pipe both the standard output and standard error from one program to another.

The pipes used by OS-9 are “unnamed pipes” and “named pipe”s.

\textbf{Unnamed Pipes}

Unnamed pipes are created by the shell when an input line with one or more exclamation point (\texttt{!}) separators is processed. For each exclamation point, the standard output of the program named to the left of the exclamation point is redirected by a pipe to the standard input of the program named to the right of the exclamation point. Individual pipes are created for each exclamation point present. For example:

$\texttt{update <master\_file \ !\ sort \ !!!\ write\_report \ >/p}$

In this example, the input for the program \texttt{update} is redirected from \texttt{master\_file.update} to the standard input for the program \texttt{sort}. The standard and error output from \texttt{sort}, in turn, become the standard input for the program \texttt{write\_report}. Standard output from \texttt{write\_report} is redirected to the printer.
Named Pipes

Named pipes are similar to unnamed pipes with one exception: a named pipe works as a holding buffer that can be opened by another process at a different time.

Named pipes are created by re-directing output to /pipe/<file>, where <file> is any legal OS-9 file name. For example:

```
$ list letters >/pipe/letters &
```

The output from the `list` command is redirected into a named pipe, /pipe/letters. The information remains in the pipe until it is listed, copied, deleted, or used in some other manner.

In Figure 4-2 the output from the command `list letters` is redirected to the named pipe, /pipe/letters. The pipe /pipe/letters remains open until the contents are used in some way. In this example, another user could later copy `letters` from the pipe into a file in their own directory by typing a command such as:

```
copy /pipe/letters /h0/usr/me/letters
```

Once the file /pipe/letters is copied, the named pipe is deleted.

![Figure 4-2. Named Pipes](image)

You can also create named pipes by writing to the named pipe from a program. Named pipes are similar to mass-storage files, except for the limitation to their size. Named pipes have attributes and owners. They may be deleted, copied, or listed using the same syntax you would use to delete, copy, or list a file. You may change the attributes of a named pipe just as you would change the attributes of a file.
dir works with /pipe. This displays all named pipes in existence. A dir -e command may be deceiving. If a named pipe is created by any utility other than copy, the default pipe size equals 128 bytes. copy expands the size of the pipe to the size of the file. This indicates the first 128 bytes of the output are in the named pipe. However, if the procs utility is executed, you see a path remains open to /pipe. If you were to copy or list the pipe, for example, the pipe continues to receive input and passes it to its output path until the input process is finished. When the pipe is empty, the named pipe is deleted automatically.

Some of the most useful applications of pipelines are character set conversion, data compression/decompression, and text file formatting. Programs designed to process data as components of a pipeline are often called filters.

**Command Grouping**

You can enclose sections of shell input lines in parentheses. This enables you to apply modifier and separators to an entire set of programs. The shell processes them by calling itself recursively as a new process to execute the enclosed program list. For example, the following commands produce the same result:

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

However, one subtle difference exists. The printer is continuously controlled by one user in the first example, while in the second case, another user could access the printer between the dir commands.

You can use command grouping to execute a group of programs sequentially with respect to each other and concurrently with respect to the shell that initiated them. For example:

```
$ (del *.backup; list stuff_* >/p)&
```

This command begins to sequentially delete all files ending in *.backup and then list to the printer the contents of any files starting with stuff_. At the same time a $ prompt appears, indicating the shell is waiting for a new command.

A useful extension of this form is to construct pipelines consisting of sequential and/or concurrent programs. For example:

```
$ (dir CMDS; dir SYS) ! makeuppercase ! transmit
This command line outputs the `dir` listings of CMDS and SYS, in that order, through a pipe to the program `makeuppercase`. The total output from `makeuppercase` is then piped to the program `transmit`.

It is important to remember that OS-9 processes commands from left to right. In the following example, the `dir` command is executed first, followed by the `procs` and `del` commands located inside the parentheses.

```
$ dir& (procs; del whatever)
```

### Shell Procedure Files

A procedure file is a text file containing one or more command lines that are identical to command lines manually entered from the keyboard. The shell executes each command line in the exact sequence given in the procedure file.

A simple procedure file might consist of `dir` on one line and `date` on another. When the name of this procedure file is entered from the command line, `dir` is run followed by `date`.

Procedure files have a number of valuable applications:

- eliminating repetitive manual entry of commonly used command sequences
- enabling the computer to execute a lengthy series of programs in the background while the computer is unattended or while you are running other programs in the foreground
- initializing your environment when you first login

You can run procedure file in the background by adding the `&` operator:

```
$ procfile&
+4
```

If a procedure file is run in the background, it should not contain any terminal I/O. Any terminal I/O caused by a background procedure file will minimally cause two or more processes try to control the same I/O path.
Notice the +4 returned by the shell in the example above. This is the process number assigned to the shell running `procfile`. The same effect could be achieved by using the `<control>C interrupt:

```
$ procfile
[<control>C is typed]
+4
```

Using `<Control>C` to place a procedure in the background only works if the procedure has not yet performed I/O to the terminal. Another limitation of the `<Control>C` interrupt occurs when the shell has not had time to set up the command for execution. If the shell has not loaded files from the disk, established pipelines, or completed other set-up activities the `<Control>C` causes the shell to abort the operation and return the shell prompt. For this reason, it is usually better to use the ampersand to place a procedure in the background.

OS-9 does not have any limit on the number of procedure files that can be simultaneously executed as long as memory is available.

Procedure files themselves can cause sequential or concurrent execution of additional procedure files.

**Using Parameters with Procedure Files**

The shell allows you to pass as many parameters as you wish to a procedure file. These parameters are entered on the command line and replace the variables located within the procedure file.

For example, if you have a procedure file, `files`, you can list the first parameter and delete the second parameter:

```
$ list files
list $(P0)
del $(P1)
```

When you enter `files` and two filenames, the first filename replaces `$P0` and the second replaces `$P1`:

```
files starter update
```

This command lists the file `starter` to your terminal screen and deletes `update`. 
If you add a third filename to the command line, it is ignored unless the variable $(P2)$ is added to the procedure file. If there is a variable $(P2)$, the third parameter is recognized and used.

The $(P*)$ variable is a concatenation of all the parameters passed to the procedure file. The following example shows a procedure file using the $(P*)$ variable and printing out the environment within the shell.

```
[7]POS: build listfil
? list $(P*)
? printenv
?
[8]POS: listfil data1 data2 data3
This is the first file Contents of data1
This is the second file Contents of data2
This is the third file Contents of data3
PORT=/pks01
HOME=/h0/USR/ROBB
SHELL=shell
USER=robb
PATH=/h0/cmds
TERM=kt7
_sh=1
PROMPT=POS:
P0=data1 First parameter
P1=data2 Second parameter
P2=data3 Third parameter
P*=data1 data2 data3 Value of variable P*
PN=3 Number of parameters passed to file listfil
```

The shell uses the PN variable to keep track of the number of parameters passed to any given procedure file.

When the procedure file has finished executing, the shell environment returns to its previous state. The variables are not passed from the procedure file back to the shell.

Do not use setenv to set variables such as P0, and P1 as they are not passed between the shell and the procedure file.
Using profile When Running Procedure Files

Typically, when a procedure file is executed, a new shell is forked to process the procedure file. Any changes affecting the shell (such as changing any of the current directories or changing the shell environment) made from within a procedure file do not affect the environment of the shell from which the procedure file was called.

The profile built-in shell command executes a procedure file without forking a child shell. This makes it possible to change current directory and environment variables from within a procedure file. For example, if you frequently work on a project located in directory /h0/USR/PROJ/MYPROJ and you want the environment variable FRAME to equal pickone whenever you work on your project, you could have a procedure file similar to the following:

```bash
$ list myproject
chd /h0/usr/proj/myproj
setenv FRAME pickone
```

When you want to work on your project, type the following command:

```
profile myproject
```

You current data directory is /h0/USR/PROJ/MYPROJ and FRAME is set to pickone. You may still pass parameter to procedure file by using profile.

The Login Shell and Special Procedure Files

The login shell is the initial shell created by the login sequence to process the user input command after logging in. To use these files, they must be located in your home directory.

`login` is processed each time the `login` command is executed. This allows you to run a number of initializing commands without remembering each and every command. `login` is processed as a command file by the login shell immediately after successfully logging on to a system. After all commands in the `login` file are processed, the shell prompts you for more commands. The main difference in handling `login` is the login shell itself actually executes the command rather than creating another shell to execute the commands.
It is possible to issue commands such as `set` and `setenv` within `.login` and have them affect the login shell. This is especially useful for setting up the environment variables `MDHOME`, `MDPATH`, `PATH`, `PROMPT`, `TERM`, and `_sh`.

Below is an example `.login` file:

```
setenv PATH ..:/h0/cmds:/d0/cmds:/dd/cmds:/h0/doc/spex
setenv PROMPT "@what next: "
setenv _sh 0
setenv TERM abm85h
setenv MDHOME querymail
date
dir
```

`.logout` is processed when `logout` is executed to exit the login shell and leave the system. `.logout` is processed before the login shell terminates. `logout` only processes the `.logout` file when given to the login shell; subsequent shells simply terminate. You could use this to execute any clean up procedures you do on a regular schedule. This might be anything from instigating a backup procedure of some sort to printing a reminder of things to do.

Below is an example `.logout` file:

```
procs
wait
echo "all processes terminated"
* basic program to instigate backup if necessary *
disk_backup
echo "backup complete"
```

**Using assign When Running Procedure Files**

The OS-9 shell allows you to assign command and strings to a single word, or assignment, for command line substitution. For example, if you prefer to use the command `cd` instead of `chd`, enter the following command line:

```
assign cd chd
```
You can also assign strings to a single word. For example, if you frequently copy a number of large files, assign the string `copy -b=50` to `copylg`:

```
assign copylg "copy -b=50"
```

You must place strings of text containing blanks in double quotes.

To find out what assignments you have already made, enter `assign` with no parameter:

```
$ assign
cd                 chd
copylg             copy -b=50
```

To remove an assignment, enter `unassign` and the assignment(s) you wish to remove:

```
unassign cd
```

`unassign` does not report errors.

By default, your assignments are kept in your environment list. This allows them to be passed from shell to shell. If you do not want your assignments to be kept in your environment list, use the `-nq` shell option. The assignments are still passed to any procedure file forked by the shell, but they are not available to the child shells.

Assignments can be used in procedure files. For example, you can set up a procedure file to copy several large files from one directory to another. You could use `copylg`, which you previously assigned. However, if someone else uses your procedure file, they may not have a `copylg` assignment, or they may have it assigned to something else. Therefore, you can `unassign copylg` and re-assign it within your procedure file. Assignments made within a procedure file are not passed back to the parent shell.

**Time-Sharing System Startup Procedure File**

OS-9 systems used for timesharing usually have a procedure file that brings the system up by means of one simple command or by using the system startup file. This procedure file initiates the timesharing monitor for each terminal. It begins by starting the system clock and initiating concurrent execution of a number of processes having their I/O redirected to each timesharing terminal.
tsmon is a special program that monitors terminals for activity. Typically, tsmon is executed as part of the start-up procedure when the system is first brought up and remains active until the system shuts down.

Tsmon is normally used to monitor I/O devices capable of bi-directional communication, such as CRT terminals. However, tsmon may also be used to monitor a named pipe. If this is done, tsmon creates the named pipe and then waits for data to be written to it by some other process.

It is possible to run several tsmon processes concurrently, each one watching a different group of devices. Because tsmon can monitor up to 28 device name pathlists, multiple tsmon processes must be run when more than 28 devices are to be monitored. Multiple tsmon processes can be useful for other reasons. For example, it may be desirable to keep modems or terminals suspected of hardware trouble isolated from other devices in the system.

Below is a sample procedure file for a timesharing system with terminals named term, t1, t2, t3, and t71:

* system startup procedure file
  echo Please Enter the Date and Time
  setime </term
  tsmon /t1 /t2 /t3&
  tsmon /t71      * This terminal has been misbehaving

In the previous example, setime has its input redirected from the system console term. This is necessary because it would otherwise attempt to read the time information from its current standard input path which is the procedure file and not the keyboard.

This login procedure does not work until a file called /d0/SYS/password with the appropriate entries has been created.

For more information on tsmon, see Chapter 7, OS-9 System Management.
The Password File

A password file is located in the SYS directory. Each line in the file is a login entry for a user, which includes fields separated by a comma.

User name
The user name may contain up to 32 characters including spaces.
If this field is empty, any name matches.

Password
The password may contain a maximum of 32 characters including spaces. If this field is omitted, no password is required for the specified user.

Group.user ID number
Both the group and the user portion of this number may be from 0 to 65535. 0.0 is the super user. This number is used by the file security system as the system-wide user ID to identify all processes initiated by the user. The system manager should assign a unique user ID to each potential user.

Initial process priority
This number may be from 1 to 65535. It indicates the priority of the initial process.

Initial execution directory
This field is usually set to /d0/CMDS. Specifying a period (.) for this field defaults to the current execution directory.

Initial data directory
This is usually the specific user directory. Specifying a period (.) for this directory defaults to the current directory.

Initial Program
This field contains the name and parameter of the program to be initially executed. This is usually shell.

Fields left empty are indicated by two consecutive commas.

The following is a sample password file:

superuser,secret,0.0,255,.,.,shell -p="@howdy"
suzy,morning,1.5,128,.,/d0/SUZY,shell
paul,dragon,3.10,100,.,/d0/PAUL,Basic
Creating a Temporary Procedure File

To perform tasks requiring a sequence of commands, you can create temporary procedure files. The cfp utility creates a temporary procedure file in the current data directory and calls the shell to execute it. After the task has been completed, cfp automatically deletes the procedure file unless you use the -nd option to specify you do not want the procedure file deleted.

The following is the syntax for the cfp utility:

```
  cfp [<opts>] [<path1>] {<path2>}
```

To use the cfp utility, type cfp, the name of the procedure file (<path1>), and the file(s) (<path2>) to be used by the procedure file. The name of the procedure file may be omitted if the -s=<string> option is used.

All occurrences of an asterisk (*) in the procedure file are replaced by the given pathlist(s) unless preceded by the tilde character (~). For example, ~* translates to *. The command procedure is not executed until all input files have been read.

For example, if you have a procedure file in your current data directory called copyit consisting of a single command line: copy *, you could put all of your C programs from two directories, PROGMS and MISC.JUNK, into your current data directory by typing the following command:

```
$ cfp copyit ../progms/*.c ../misc.junk/*.c
```

If you do not have a procedure file, you can use the -s option. The -s option causes the cfp utility to read the string surrounded by quotes instead of a procedure file. For example:

```
$ cfp "-s=copy *" ../progms/*.c ../misc.junk/*.c
```

In this case, the cfp utility creates a temporary procedure file to copy every file ending in .c in both PROGMS and MISC.JUNK to the current data directory. The procedure file created by cfp is deleted when all the files have been copied.
Using the \(-s\) option is convenient because you do not have to edit the procedure file if you want to change the copy procedure. For example, if you are copying large C programs, you may want to increase the memory allocation to speed up the process. You could allocate the additional memory on the \texttt{cfp} command line:

\[
\texttt{\$ cfp "-s=copy -b100 *" ../progms/*.c ../misc.junk/*.c}
\]

You can use the \(-z\) and \(-z=<\text{file}>\) options to read the file names from either standard input or a file. The \(-z\) option is used to read the file names from standard input. For example, if you have a procedure file called \texttt{count.em} containing the command \texttt{count -l *} and you want to count the lines in each program to see how large the programs are before you copy them, you could type the following command line:

\[
\texttt{\$ cfp -z count.em}
\]

The command line prompt does not appear because the \texttt{cfp} utility is waiting for input. Type in the file names on separate command lines. For example:

\[
\texttt{\$ cfp -z count.em}
\]

\[
\texttt{../progms/*.c}
\]

\[
\texttt{../misc.junk/*.c}
\]

When you have finished typing the file names, press the carriage return a second time to get the shell prompt.

\[
\text{For more information on \texttt{cfp}, see the Utilities Reference manual.}
\]

If you have a file containing a list of the files you want copied, you could type the following command:

\[
\texttt{\$ cfp -z=files "-s=copy *"}
\]
Multiple Shells

Like all OS-9 utilities, the shell can be simultaneously executed by more than one process. This means in addition to all users having their own shells, an individual user can have multiple shells.

New shells can be created with the procedure file. For example, to execute a shell whose standard input is obtained from procfile, type the following command:

```bash
$ shell <procfile
```

The new shell automatically accepts and executes the command lines from the procedure file instead of a terminal keyboard. This technique is sometimes called batch processing.

Shells can also fork new shells by simply processing the procedure file:

```bash
$ procfile
```

Basically, both of the above commands execute the commands found in the procfile file. By creating new shells, you can also move around the file system more efficiently. To demonstrate this application use the sample directory system in Figure 4-3.

### Figure 4-3. An Example Directory

If your current data directory is DIR_9 and you want to work on file_8, you could change your current data directory to DIR_8 and access the file by typing the following command:

```bash
chd /d0/DIRECTORY_3/DIR_8
```
To return to DIR_9 you execute a similar command. This is somewhat inconvenient and involves always knowing the path to each directory.

Instead, you can create a shell and change directories:

```
$ (chd /d0/DIRECTORY_3/DIR_8)
```

This makes your current directory DIR_8, but you can return to DIR_9 by pressing the <Escape> (Esc) key. By this method, you may use any directory as a base directory and fork a shell out to any other directory.

You may continue to imbed as many shells as you like. Each time you press the <Escape> key, you are taken to the previous shell. In this fashion you could conceivably escape from DIRECTORY_2 to DIR_8 to DIR_6 to DIR_9.

You should experiment with the multiple shell aspects to fully use OS-9.

Because of the nature of jumping from shell to shell, it is easy to get lost. `pd` displays a complete pathlist from the root directory to your current data directory.

Likewise, when running multiple shells, it is easy to forget how many shells are running. If the `_sh` environment variable is set to 1 and the shell prompt includes an “at” sign (@), the number of shells replaces the @ in the prompt. For example, if three shells are being run under each other and the history count is on, the prompt might look like the following example:

```
3.[5]now what:
```

### The procs Utility

Because OS-9 is a multi-tasking operating system, you often have more than one process executing at a time. The `procs` utility displays a list of processes running on the system you own. This allows you to keep track of your current processes.

Processes can switch states rapidly, usually many times per second. Therefore the procs display is a snapshot taken at the instant the command is executed and shows only those processes running at that exact moment.
The `procs` command displays ten items of information for each process:

### Table 4-13. Information From `procs`

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>The process ID</td>
</tr>
<tr>
<td>PId</td>
<td>The parent process ID</td>
</tr>
<tr>
<td>Grp.usr</td>
<td>The group and user number of the owner of the process</td>
</tr>
<tr>
<td>Prior</td>
<td>The initial priority of the process</td>
</tr>
<tr>
<td>MemSiz</td>
<td>The amount of memory the process is using</td>
</tr>
<tr>
<td>Sig</td>
<td>The number of any pending signals for the process</td>
</tr>
<tr>
<td>S</td>
<td>State of the process</td>
</tr>
</tbody>
</table>

- **CPU** = Process is currently in the CPU. This will always be the `procs` command since it has to be running when it takes the snapshot of the process table.

- **a** = Active. Process wants CPU time, but must wait because another process is in the CPU.

- **d** = Debug. Process is currently being debugged.

- **e** = Event. Process is blocked waiting on an event.

- **p** = Semaphore. Process is blocked waiting on semaphore.

- **s** = Sleeping. Process is blocked waiting on a signal or time value to elapse.

- **w** = Waiting. Process is waiting on a child process to terminate.

- **z** = Suspended. Process is blocked by a kernel system call.

- **-** = Zombie. Process has been terminated, but the parent has not performed a wait to read the exit status.

<table>
<thead>
<tr>
<th>P</th>
<th>Wait on semaphore</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Time</td>
<td>The amount of CPU time the process has used</td>
</tr>
</tbody>
</table>
Table 4-13. Information From procs (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>The elapsed time since the process started</td>
</tr>
<tr>
<td>Module &amp; I/O</td>
<td>The process name and standard I/O paths:</td>
</tr>
<tr>
<td></td>
<td>&lt; = standard input</td>
</tr>
<tr>
<td></td>
<td>&gt; = standard output</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt;&gt; = standard error output</td>
</tr>
<tr>
<td></td>
<td>If several of the paths point to the same pathlist, the identifiers for the paths are merged.</td>
</tr>
</tbody>
</table>

Below is an example of procs:

```
$ procs
Id  PID Grp.Usr Prior MemSiz Sig  S   CPU Time Age Module & I/O
2   1  22.150   128   0.25k  0 w    0.01   ??? sysgo <<>>term
3   3  22.150   128   4.75k  0 w    4.11  1:13 shell <<>>term
4   3  22.150   128    5.00k  0 a   12:42.06 0.14 xhog <<>>term
5   0  22.150   128   8.50k  0 *    0.08  0:00 procs <<>>term
6   0  22.150   128    4.00k  0 s    0.02  1:12 tsmon <<>>t1
7   0  22.150   128    4.00k  0 s    0.01  1:12 tsmon <<>>t2
```

`procs -a` displays the process ID, the parent process ID, the process name and standard I/O paths, and six new pieces of information:

Table 4-14. Information From procs -a

<table>
<thead>
<tr>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aging</td>
<td>The age of the process based on the initial priority and how long it has waited for processing</td>
</tr>
<tr>
<td>F$calls</td>
<td>The number of service request calls made</td>
</tr>
<tr>
<td>I$calls</td>
<td>The number of I/O requests made</td>
</tr>
<tr>
<td>Last</td>
<td>The last system call made</td>
</tr>
<tr>
<td>Read</td>
<td>The number of bytes read</td>
</tr>
<tr>
<td>Written</td>
<td>The number of bytes written</td>
</tr>
</tbody>
</table>
The following is an example of procs -a:

```
$ procs -a
Id PId Aging F$calls I$calls Last   Read  Written Module & I/O
2   1   129    5        1 Wait         0     0       sysgo <<<term
3   2   132    116      127 Wait       282   129     shell <<<term
4   3   11    1        0 TLink        0     0       xhog <<<term
5   3   128    7        4 GPrDsc       0     0       procs <<<term
6   0   130    2        7 ReadLn       0     0       tsmon <<<t1
7   0   129    2        7 ReadLn       0     0       tsmon <<<t2
```

The -b option displays all information from procs and procs -a. The -e option displays information for all processes in the system.

### Waiting for Background Procedures

If the multi-tasking ability of OS-9 is used, there are times when a number of procedures are running in the background. If it is important to wait for these tasks to finish before running a new procedure, use the `w` or `wait` built-in shell command.

- `w` wait for the last child process to be executed to finish.
- `wait` wait for all child processes running in the background to finish.
- A child process is a process being executed by the current shell or a child of the shell.
- `wait` does not wait until a process forked with the plus sign (+) concurrent execution separator finishes execution. Processes forked with the plus sign are orphan processes.

For example, to create a document from three different files, where each file has to be sorted by different fields, use the following files:

*start of first procedure file*

```bash
qsort -f=1 file1&
qsort -f=2 file2&
qsort -f=3 file3&
wait
merge file1 file2 file3 >report
```

*start of second procedure file*

```bash
qsort -f=1 file1
qsort -f=2 file2
qsort -f=3 file3
merge file1 file2 file3 >report
```

The first procedure file is faster, as each file is processed concurrently.
Stopping Procedures

You can use two methods to stop a procedure. The first method involves the `<Control>C` or `<Control>E` signal. The second method uses the `kill` utility.

- `<Control>C` stops the shell from waiting for the process to terminate and returns a prompt for a command.
- `<Control>E` forwards the keyboard abort signal to the process and immediately prompts for input.

The shell handles these keyboard generated signals in the following manner. If either of these signals are received while the shell is waiting for keyboard input the following messages are issued:

```
$ Read I/O error - Error #000:177   [ ^E typed ]
$ Read I/O error - Error #000:177   [ ^C typed ]
```

These are the standard messages given whenever an I/O error occurs when reading command input data.

If the shell is waiting for keyboard input and `<Control>E` is typed, the shell forwards the keyboard abort signal to the current process and immediately prompts for command input:

```
$ sleep 500
[ ^E is typed]
abort
$
```

The `abort` message is typed by the shell to acknowledge receipt of the interrupt.

If the shell is waiting for keyboard input and you enter `<Control>C`, the shell stops waiting for the current process to terminate and prompts for command input. This action is similar to using an ampersand on the command line. For example:

```
$ sleep 500
[ ^C is typed]
+8
$
```
It is important to remember that using \texttt{<Control>C} in this fashion is possible only if the command in question has not yet performed I/O to the terminal. The signal is only received by the last process to perform I/O. If the shell has not yet finished setting up the command for execution, the signal causes the shell to abort the operation and returns the prompt.

You must own the procedure or be the super user to kill a specified process.

You can also use the \texttt{kill} utility to terminate background processes by specifying the process number of the process to be killed. Obtain the process number of the process from \texttt{procs}. Use the \texttt{kill} utility in the following manner:

\texttt{kill <proc num>}

For example, if you want to terminate a process called \texttt{xhog}, you would first execute a \texttt{procs}:

\begin{verbatim}
$ procs
    Id  PId  Grp.Usr  Prior  MemSiz  Sig S   CPU Time  Age
Module & I/O
  3   2   7.03   128    4.75k   0 w       4.11 01:13
shell <>>>term
  4   3   7.03     5    4.00k   0 a   12:42.06 00:14
xhog <>>>term
  5   3   7.03   128    8.50k   0 *   0.08 00:00
procs <>>term
\end{verbatim}

From \texttt{procs}, you can see the process number for \texttt{xhog} is four. You can then type the following command:

\begin{verbatim}
$ kill 4
\end{verbatim}

When you execute \texttt{procs} again, you find \texttt{xhog} is no longer shown.

To use the \texttt{kill} utility, complete the following steps:

\begin{itemize}
  \item \textbf{Step 1.} Use the \texttt{procs} utility to get the process number.
  \item \textbf{Step 2.} Type \texttt{kill <proc num>}.\end{itemize}
Either of these methods terminates any process running in the background with one exception: if a process is waiting for I/O, it may not die until the current I/O operation is complete. Therefore, if you terminate a process and procs shows it still exists, it is probably waiting for the output buffer to be flushed before it can die.

## Command History

As you enter command lines, the commands are saved in a buffer. This is a history of your commands. To see the commands you have entered, type `hist` on the command line:

```
[5]$ hist
Shell History
----------
1) mkdir /h0/usr/TMS
2) chd /h0/usr/tms
3) build stat
4) procs
5) hist
[6]$ 
```

These commands may be re-executed or retrieved using tildes (~). One tilde followed by a number (~<num>) executes the command pointed to by <num>. For example, entering ~4 on the command line causes the shell to execute the fourth command in your history list.

In the example above, the `procs` command is executed:

```
[6]$ ~4
   Id PID Grp.Usr Prior MemSiz Sig S  CPU Time   Age Module & I/O
   3   2   6.10   128    5.25k    0 w      5.02 02:34 shell
<>>>term
   4   3   6.10   128    8.50k    0 *      0.08 00:00 procs <>>term
[7]$ hist
Shell History
----------
3) build stat
4) procs
5) hist
6) procs
7) hist
[8]$ 
```

Entering `~4 -e` tells the shell to execute `procs -e`.  

You can also re-execute/retrieve commands using a tilde followed by text (~<text>). The OS-9 shell searches backwards through the history buffer for the text. For example, if you enter ~uma on the command line, the command umacs stat is executed.

You cannot include spaces in your text. Also, the text must be the first characters in the command line. In the previous example, entering ~acs would produce an error.

Entering a number after two tildes (~~<num>) places the command in the command line buffer, just as if it were the last command entered. For example, by typing ~~3, the command is placed in a buffer as if it had just been executed. By entering <Control>A, you can retrieve the command line. It is placed after the shell prompt:

```
[8]$ ~~3
[8]$ <control>A
[8]$ build stat
```

You can either execute the command by pressing a carriage return or you can edit the command line and then execute it:

```
[8]$ build stat.tst
```

In the previous example, the history number ([8]) did not change when the ~~3 command and the <Control>A were entered. The history number only changes when a command line is entered. The ~~3 causes the command to be placed in the buffer. Likewise, <Control>A causes the command to be placed on the command line. Entering blank lines also does not increase the history count.

You can also enter text after two tildes (~~<text>). For example, you could type ~~uma. Then enter <Control>A to retrieve the command. Once it appears on the command line, you can edit it.

### Error Reporting

Many programs, including the shell, use the OS-9 standard error reporting function. This displays a brief description of the error and an error number on the standard error path.

If an expanded error description is desired, set the -e and the -v shell options. This prints error messages from /dd/SYS/errmsg on standard output.
This chapter explains the make utility in detail. The following sections are included:

- The make Utility
- Examples Using the make Utility
The make Utility

Many types of files are dependent upon various other files in their creation. If the files comprising the final product are updated, the final product becomes out-of-date. The **make** utility is designed to automate the maintenance and recreation of files that change over time.

**make** maintains files by using a special type of procedure file known as a “makefile”. The makefile describes the relationship between the final product and the files comprising the final product. For the purpose of this discussion, the final product is referred to as the “target file” and the files comprising the target file are referred to as “dependents”.

A makefile contains three types of entries:

- dependency entries
- command entries
- comment entries

A dependency entry specifies the relationship of a target file and the dependents used to build the target file. The entry has the following syntax:

```
<target>: [[<dependent>],<dependent>]
```

The list of files following the target file is known as the dependency list. Any number of dependents can be listed in the dependency list. Any number of dependency entries can be listed in a makefile. A dependent in one entry may also be a target file in another entry. However, there is only one main target file in each makefile. The main target file is usually specified in the first dependency entry in the makefile.

If necessary, a command entry specifies the command that was executed to update a particular target file. **make** updates a target file only if its dependents are newer than itself. If no instructions for update are provided, **make** attempts to create a command entry to perform the operation.

**make** recognizes a command entry via a line beginning with one or more spaces or tabs. Any legal OS-9 command line is acceptable. More than one command entry can be given for any dependency entry. Each command entry line is assumed to be complete unless it is continued from the previous command with a backslash (\). Comments should not be interspersed with commands.
For example:

```
<target>: [[<file>],<file>]
<OS-9 command line>
<OS-9 command line>\
<continued command line>
```

A comment entry consists of any line beginning with an asterisk (*). All characters following a pound sign (#) are also ignored as comments unless a digit immediately follows the pound sign. In this case, the pound sign is considered part of the command entry. All blank lines are ignored. For example:

```
<target>: [[<file>],<file>]

* the following command will be executed if the
* dependent files are newer than the target file
<OS-9 command line> # this is also a comment
```

Spaces and tabs preceding non-command continuation lines are ignored.

You can continue any entry on the following line by placing a space followed by a backslash (\) at the end of the line to be continued. All entries longer than 256 characters must be continued on another line. All continuation lines must adhere to the rules for its type of entry. For example, if a command line is continued on a second line, the second line must begin with a space or a tab:

```
FILE: aaa.r bbb.r ccc.r ddd.r eee.r \  
fff.r ggg.r

    touch aaa.r bbb.r ccc.r \  
    ddd.r eee.r fff.r ggg.r
```
Running the Make Utility

To run the `make` utility, type `make`, followed by the name of the file(s) to create and any options desired.

`make` processes the makefile three times.

- During the first pass, `make` examines the makefile and sets up a table of dependencies. This table of dependencies stores the target file and the dependency files exactly as they are listed in the makefile. When `make` encounters a name on the left side of a colon, it first checks to see if it has encountered the name before. If it has, `make` connects the lists and continues.

- After reading the makefile, `make` determines the target file on the list. It then makes a second pass through the dependency table. During this pass, `make` tries to resolve any existing implicit dependencies. Implicit dependencies are discussed below.

- `make` does a third pass through the list to get and compare the file dates. When `make` finds a file in a dependency list that is newer than its target file, it executes the specified command(s). If no command entry is specified, `make` generates a command based on the assumptions given in the next section. Because OS-9 only stores the time down to the closest minute, `make` remakes a file if its date matches one of its dependents.

When a command is executed, it is echoed to standard output. `make` normally stops if an error code is returned when a command line is executed.

To understand the relationship of the target file, its dependents and the commands necessary to update the target file, the structure of the makefile must be carefully examined.

Implicit Definitions

Any time a command line is generated, `make` assumes the target file is a program to compile. Therefore if the target file is not a program to compile, any necessary command entries must be specified for each dependency list. `make` uses the following definitions and rules when forced to create a command line.

- **object files** Files with no suffixes. An object file is made from a file capable of relocation and linked when it needs to be made.
<table>
<thead>
<tr>
<th>Relocatable files</th>
<th>Files appended by the suffix: .r. Relocatable files are made from source files and are assembled or compiled if they need to be made.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source files</td>
<td>Files having one of the following suffixes: .a, .c, .f, or .p.</td>
</tr>
<tr>
<td>Default compiler</td>
<td><code>cc</code></td>
</tr>
<tr>
<td>Default assembler</td>
<td>The default options are processor specific; some examples include:</td>
</tr>
<tr>
<td></td>
<td>• appc for PowerPC processors</td>
</tr>
<tr>
<td></td>
<td>• a386 for 80386 processors</td>
</tr>
<tr>
<td>Default linker</td>
<td><code>cc</code></td>
</tr>
</tbody>
</table>

**Macro Recognition**

In addition to recognizing compilation rules and definitions, `make` recognizes a macro by the dollar sign (\$) character in front of the name. If a macro name is longer than a single character, the entire name must be surrounded by parentheses. For example, \$R refers to the macro R, \$(PFLAGS) refers to the macro PFLAGS, \$(B) and \$B refer to the macro B, and \$BR is interpreted as the value for the macro B followed by the character R. You may place macros in the makefile for convenience or on the command line for flexibility. Macros are allowed in the form of `<macro name> = <expansion>`. The expansion is substituted for the macro name whenever the macro name appears.

Defining a macro in a command line macro overrides the macro definition in a makefile. To increase `make`’s flexibility, you can define special macros in the makefile. `make` uses these macros when assumptions must be made in generating command lines or when searching for unspecified file. For example, if no source file is specified for `program.r`, make searches either the directory specified by `SDIR` or the current data directory for `program.a` (or `.c, .p, .f`).
make recognizes the following special macros:

**Table 5-1. make Macros**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC=&lt;comp&gt;</td>
<td>make uses this compiler when generating command lines. The default is cc.</td>
</tr>
<tr>
<td>CFLAGS=&lt;opts&gt;</td>
<td>These compiler options are used in any necessary compiler command lines.</td>
</tr>
<tr>
<td>LC=&lt;link&gt;</td>
<td>make uses this linker when generating command lines. The default is cc.</td>
</tr>
<tr>
<td>LFLAGS=&lt;opts&gt;</td>
<td>These linker options are used in any necessary linker command lines.</td>
</tr>
<tr>
<td>ODIR=&lt;path&gt;</td>
<td>make searches the directory specified by &lt;path&gt; for all files with no suffix or relative pathlist. If ODIR is not defined in the makefile, make searches the current directory by default.</td>
</tr>
<tr>
<td>RC=&lt;asm&gt;</td>
<td>make uses this assembler when generating command lines. The default for users on 680x0 processors is r68. The default for users on x86 processors is cc.</td>
</tr>
<tr>
<td>RDIR=&lt;path&gt;</td>
<td>make searches the directory specified by &lt;path&gt; for all relocatable files not specified by a full pathlist. If RDIR is not defined, make searches the current directory by default.</td>
</tr>
<tr>
<td>RFLAGS=&lt;opts&gt;</td>
<td>These assembler options are used in any necessary assembler command lines.</td>
</tr>
<tr>
<td>SDIR=&lt;path&gt;</td>
<td>make searches the directory specified by &lt;path&gt; for all source files not specified by a full pathlist. If SDIR is not defined in the makefile, make searches the current directory by default.</td>
</tr>
</tbody>
</table>
Some reserved macros are expanded when a command line associated with a particular file dependency is forked. You may use these macros only on a command line. When you need to be explicit about a command line but have a target program with several dependencies, these macros are useful. In practice, they are wildcards with the following meanings:

Table 5-2. make Wildcards

<table>
<thead>
<tr>
<th>Macro</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$@</td>
<td>expands to the file name made by the command</td>
</tr>
<tr>
<td>$*</td>
<td>expands to the prefix of the file to be made</td>
</tr>
<tr>
<td>$?</td>
<td>expands to the list of files found to be newer than the target on a given dependency line</td>
</tr>
</tbody>
</table>

make Generated Command Lines

make is capable of generating three types of command lines: compiler command lines, assembler command lines, and linker command lines.

- Compiler command lines are generated if a source file with a suffix of .c, .p or .f needs to be recompiled. The compiler command line generated by make has the following syntax:
  \[ $(CC) $(CFLAGS) -r=$(RDIR) $(SDIR)/<file>[.c, .f, or .p] \]

- Assembler command lines are generated when an assembly language source file needs to be re-assembled. The assembler command line generated by make has the following syntax:
  \[ $(RC) $(RFLAGS) $(SDIR)/<file>.a -o=$(RDIR)/<file>.r \]

- Linker command lines are generated if an object file needs to be relinked in order to re-make the program module. The linker command line generated by make has the following syntax:
  \[ $(LC) $(LFLAGS) $(RELS)/<file>.r -f=$(ODIR)/<file> \]

When make is generating a command line for the linker, it looks at its list and uses the first relocatable file it finds, but only the first one. For example:

prog: x.r y.r z.r

generates:

cc x.r, not cc x.r y.r z.r or cc prog.r
**make Options**

Several options allow make even greater versatility for maintaining files/modules. You can include these options on the command line in the makefile or when you run make.

When a command is executed, it is echoed to standard output unless the \(-s\), or silent, option is used or the command line starts with an “at” sign (@). When the \(-n\) option is used, the command is echoed to standard output but not actually executed. This is useful when building your original makefile.

make normally stops if an error code is returned when a command line is executed. Errors are ignored if the \(-i\) option is used or if a command line begins with a hyphen.

It is often helpful to see the file dependencies and the dates associated with each of the file in the list. The \(-d\) option turns on the make debugger and gives a complete listing of the macro definitions, a listing of the files as it checks the dependency list and all the file modification dates. If it cannot find a file to examine its date, it assumes a date of -1/00/00 00:00, indicating the necessity to update the file.

If you want to update the date on a file, but do not want to remake it, you can use the \(-t\) option. make merely opens the file for update and then closes it, thus making the date current.

If you are explicit about your makefile dependencies and do not want make to assume anything, you may use the \(-b\) option to turn off the built-in rules governing implicit file dependencies.

**Table 5-3. make Options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-?)</td>
<td>Display the usage of make.</td>
</tr>
<tr>
<td>(-b)</td>
<td>Do not use built in rules.</td>
</tr>
<tr>
<td>(-bo)</td>
<td>Do not use built in rules for object files.</td>
</tr>
<tr>
<td>(-d)</td>
<td>Print the dates of the files in makefile (debug mode).</td>
</tr>
<tr>
<td>(-dd)</td>
<td>Double debug mode. Verbose.</td>
</tr>
<tr>
<td>(-f-)</td>
<td>Read the makefile from standard input.</td>
</tr>
<tr>
<td>(-f=&lt;path&gt;)</td>
<td>Specify &lt;path&gt; as the makefile.</td>
</tr>
</tbody>
</table>

If <path> is specified as a hyphen (-), make commands are read from standard input.
Chapter 5: Making Files

**Examples Using the make Utility**

The remainder of this chapter shows details different ways to maintain programs with make.

### Updating a Document

The following example shows how `make` maintains current documentation composed of different sections:

```plaintext
utils.man: chap1 chap2 apdx
    del utils.man.old; rename utils.man utils.man.old
    merge chap1 chap2 apdx >utils.man
chap1: c1a c1b c1c c1d
    del chap1.old; rename chap1 chap1.old
    list c1a c1b c1c c1d ! lxfilter >chap1
chap2: c2a c2b c2c
    del chap2.old; rename chap2 chap2.old
    list c1a c1b c1c c1d ! lxfilter >chap1
apdx: functions header footer
    del apdx.old; rename apdx apdx.old
    qsort functions >/pipe/func
    list header /pipe/func footer ! lxfilter >apdx
```

The above makefile creates the file `utils.man`. `utils.man` is created from three files: `chap1`, `chap2`, and `apdx`. Each of these files is in turn created from the files listed in their dependency lists.

If `chap1`, `chap2`, and/or `apdx` have dependencies with a more recent date, the command following their respective dependency entries are executed. If `chap1`, `chap2`, and/or `apdx` are re-created, the commands following the initial dependency entry are executed.

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>Ignore errors.</td>
</tr>
<tr>
<td>-n</td>
<td>Do not execute commands, but does display them.</td>
</tr>
<tr>
<td>-s</td>
<td>Silent mode: execute commands without echo.</td>
</tr>
<tr>
<td>-t</td>
<td>Update the dates without executing commands.</td>
</tr>
<tr>
<td>-u</td>
<td>Do the make regardless of the dates on files.</td>
</tr>
<tr>
<td>-x</td>
<td>Use the cross-compiler/assembler.</td>
</tr>
<tr>
<td>-z</td>
<td>Read a list of make targets from standard input.</td>
</tr>
<tr>
<td>-z=&lt;path&gt;</td>
<td>Read a list of make targets from &lt;path&gt;.</td>
</tr>
</tbody>
</table>

Table 5-3. make Options (Continued)
Compiling C Programs

In this example, make is used to compile high level language modules. Each command and dependency is specified.

```
program: xxx.r yyy.r
  cc xxx.r yyy.r -xf=program
xxx.r: xxx.c /d0/defs/oskdefs.h
  cc xxx.c -r
yyy.r: yyy.c /d0/defs/oskdefs.h
  cc yyy.c -r
```

This makefile specifies program is made up of two .r files: xxx.r and yyy.r. These files are dependent upon xxx.c and yyy.c respectively and both are dependent on the oskdefs.h file.

If either xxx.c or /d0/defs/oskdefs.h has a date more recent than xxx.r, the command cc xxx.c -r is executed. If yyy.c or /d0/defs/oskdefs.h is newer than yyy.r, then cc yyy.c -r is executed. If either of the former commands are executed, the command cc xxx.r yyy.r -xf=program is also executed.

In this example, make specifies each command it must execute. Often this is unnecessary, as make uses specific definitions, macros, and built-in assumptions to facilitate program compilation and generate its own commands.

Knowing how make works and understanding the implicit rules can simplify coding immensely:

```
program: xxx.r yyy.r
  cc xxx.r yyy.r -xf=program
xxx.r yyy.r: /d0/defs/oskdefs
```

This makefile exploits the make utility awareness of file dependencies. No mention is made of the C language files; therefore, make looks in the directory specified by the macro definition SDIR = <path> and adjusts the dependency list accordingly. In this case, make also generates a command line to compile xxx.r and yyy.r if either needs updating.

Further simplification is possible if program is made up of only one source file:

```
program:
```
make assumes the following from this simple command:

- program has no suffix. It is an object file and therefore relies on relocatable files to be made.
- No dependency list is given; therefore, make creates an entry in the table for program.r.
- After creating an entry for program.r, make creates the entry for a source file connected to the relocatable file.

Assuming it found program.a, make checks the dates on the various files and generates one or both of the following commands if required:

```
appc program.a -o=program.r
cc program.r -f=program
```

### Using Macros

Using these inherent features of make is especially helpful if you have several object files you want make to check:

```
* beginning
ODIR = /d0/cmds
RDIR = rels
UTILS = attr copy load dir backup dsave
SDIR = ../utils/sources
utils.files: $(UTILS)
    touch utils.files
* end
```

make searches rels for the .r files (attr.r, copy.r, and so on) and looks in ../utils/sources for the .c files named in the UTILS= line. make then generates the proper commands to compile and/or link any of the programs needing to be made. If one of the files in UTILS is made, the command touch utils.files is forked to maintain a current overall date.
Creating make

The following example is a makefile to create `make`:

* beginning
ODIR = /h0/cmds
RDIR = rels
CFILES = domake.c doname.c dodate.c domac.c
RFILES = domake.r doname.r dodate.r
PFLAGS = -p64 -nh1
R2 = ../test/domac.r
RFLAGS = -q
make: $(RFILES) $(R2) getfd.r
linker
$(RFILES): defs.h
$(R2): defs.h
  cc $*.c -r=../test
print.file: $(CFILES)
  pr $? $(PFLAGS) >/p1
  touch print.file
*end

The makefile in this example looks for the `.r` files listed in `RFILES in the directory specified by RDIR: rels. The only exception is `../test/domac.r`, which has a complete pathlist specified.

Even though `getfd.r` does not have any explicit dependents, its dependency on `getfd.a` is still checked. The source files are all found in the current directory.

This makefile can also be used to make listings. By typing `make print.file` on the command line, make expands the macro `$?` to include all of the files updated since the last time `print.file` was updated. If you keep a dummy file called `print.file` in your directory, make only prints out the newly made files. If no `print.file` exists, all files are printed.
Performing Backups

This chapter explains the concept of incremental backups. The OS-9 utilities that create backups are detailed here. This chapter also offers two different strategies for making backups. The following sections are included:

- Incremental Backups
- Using the tape Utility
- Backing Up the System Disk
Incremental Backups

Incremental backups save significant time and storage space compared to full system backups. Incremental backups save only the files that have changed since the last backup. A full system backup must still be performed, but with the use of incremental backups a full system backup does not need to be performed very often.

OS-9 provides two utilities to facilitate incremental backups. You can use each with either tape or disk media:

- `fsave`
- `frestore`

Certain terms are important to know for the discussion of incremental backups:

- **Level 0 backup** is a common term used to denote a full system backup.

  Consequent incremental backups are referenced by different level numbers. For example, a level 5 backup includes all files changed since the most recent backup with a level less than 5. While this sounds complex, it is actually quite easy to use and extremely helpful.

- **Source device** is the directory structure or file you are backing up.

- **Target device** is the tape or disk that holds your backup information.

Making an Incremental Backup: The `fsave` Utility

The `fsave` utility performs an incremental backup of a directory structure to tape(s) or disk(s). The syntax for the `fsave` utility is shown below:

```
fsave [<opts>] [<path>]
```

Typing `fsave` alone on the command line makes a level 0 backup of the current directory onto a target device with the name `/mt0`.

// `/mt0` is the OS-9 device name for a tape device.
/**h0/sys/backup_date is a backup log file maintained by fsave. Each time an fsave is executed, the backup log is updated. The backup log keeps track of the name of the backup, the date it was created and, more importantly, the level of the backup. When fsave is executed, this backup log is examined to find the specified level of the current backup and the previous backups with the same name. Once the backup is finished, a new entry is made in the file indicating the date, name, level, and other information about the current backup.

**fsave Options**

During the discussion of the actual fsave procedure, references to fsave options are made. The options are shown in Table 6-1.

**Table 6-1. fsave Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-?</td>
<td>Display the usage of fsave.</td>
</tr>
<tr>
<td>-b[=]&lt;int&gt;</td>
<td>Allocate &lt;int&gt;k buffer size to read files from the source disk.</td>
</tr>
<tr>
<td>-d[=]&lt;dev&gt;</td>
<td>Specify the target device to store the backup. The default is /mt0.</td>
</tr>
<tr>
<td>-e</td>
<td>Do not echo file pathlists being saved to target device.</td>
</tr>
<tr>
<td>-f[=]&lt;path&gt;</td>
<td>Save to the file specified by &lt;path&gt;.</td>
</tr>
<tr>
<td>-g[=]&lt;int&gt;</td>
<td>Specify a backup of files owned by group number &lt;int&gt; only.</td>
</tr>
<tr>
<td>-l[=]&lt;int&gt;</td>
<td>Specify the level of the backup to be performed.</td>
</tr>
<tr>
<td>-m[=]&lt;path&gt;</td>
<td>Specify the pathlist of the date backup log file to be used. The default is /h0/sys/backup_dates.</td>
</tr>
<tr>
<td>-p</td>
<td>Turn off the mount volume prompt for the first volume.</td>
</tr>
<tr>
<td>-s</td>
<td>Display the pathlists of all files needing to be saved and the size of the entire backup without actually executing the backup procedure.</td>
</tr>
<tr>
<td>-t[=]&lt;dirpath&gt;</td>
<td>Specify alternate location for the temporary index file.</td>
</tr>
<tr>
<td>-u[=]&lt;int&gt;</td>
<td>Specify backup of files owned by user number &lt;int&gt;.</td>
</tr>
</tbody>
</table>
The fsave Procedure

When starting an fsave procedure, fsave prompts you to mount the first volume to use. Volume, in this case, refers to the disk or tape used to store the backup:

    fsave: please mount volume.
    (press return when mounted).

If a disk is used as the backup medium, fsave verifies the disk and displays the following information: (The numbers below are used only as an example.)

    verifying disk
    Bytes held on this disk: 546816
    Total data bytes left:   62431
    Number of Disks needed: 1

The most common error found when executing fsave is a record lock error. Record lock errors are caused when another user has the file in question open. fsave operations should only be done when no one else is using the system to prevent record lock errors. If a tape is used as the backup medium, no preliminary information is displayed and the backup begins at this point.

As each file is saved to the backup device, the file pathlist is echoed to the terminal. If this is a long backup, you may want to use the -e option to turn off the echoing of pathlists.

If fsave receives an error when trying to backup a file, it displays the following message and continues the fsave operation:

    error saving <file>, error - <error number>, it is incomplete

If the backup requires more than one volume, fsave prompts you to mount the next volume before continuing.

Table 6-1. fsave Options (Continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>Do not verify the disk volume when mounted.</td>
</tr>
<tr>
<td>-x[=]&lt;int&gt;</td>
<td>Pre-extend temporary file. &lt;int&gt;; given in kilobytes.</td>
</tr>
</tbody>
</table>
At the end of the backup, \texttt{fsave} prints the following information:

\begin{verbatim}
fsave: Saving the index structure
Logical backup name: 
Date of backup: 
Size of backup: 
Size of temp/index: 
Backup made by: 
Data bytes written: 
Number of files: 
Number of volumes: 
Index is on volume: 
\end{verbatim}

The index to the backup is saved on the last volume used.

\texttt{fsave} performs recursive backups for each pathlist if one or more directories are specified on the command line. A maximum of 32 directories may be specified on the command line.

The \texttt{-d} option allows you to specify an alternate target device. The default device is \texttt{/mt0}.

Use the \texttt{-m} option to specify an alternative backup log file. The default pathlist is \texttt{/h0/sys/backup_dates}.

Different levels of backups may be specified with the \texttt{-l} option. A higher level backup only saves files that have changed since the most recent backup with the next lower number. For example, a level 1 backup saves all files changed since the last level 0 backup.

When using disks for backup purposes, \texttt{fsave} does not use an RBF file structure to save the file on the target disk. It creates its own file structure. This makes the backup disk unusable for any purpose other than \texttt{fsave} and \texttt{frestore} without reformatting the disk.

Any data stored on the disk before use by \texttt{fsave} is destroyed by the backup.

\textbf{Example fsave Commands}

Typing \texttt{fsave} on a command line specifies a level 0 backup of the current directory. This assumes the device \texttt{/mt0} is to be used and \texttt{/h0/SYS/backup_dates} is used as the backup log file for this backup.
The following command specifies a level 2 backup of the current directory using the /mt1 device. /h0/misc/my_dates is used as the backup log file:

$ fsave -l=2 -d=/mt1 -m=/h0/misc/my_dates

The following command specifies a level 0 backup of all files owned by user 0.0 in the CMDS directory, if CMDS is in your current directory:

$ fsave -pb=32 -g=0 -u=0 -d=/d2 CMDS

This backup uses /d2 as the target device and /h0/sys/backup_dates as the backup log file. The mount volume prompt is not generated for the first volume. A 32k buffer is used to read the files from the CMDS directory.

Restoring Incremental Backups: The frestore Utility

The frestore utility restores a directory structure from multiple volumes of tape or disk media. The syntax for the frestore utility is listed below:

frestore [<opts>] [<path>]

Typing frestore on the command line attempts to restore a directory structure from the device /mt0 to the current directory.

Specifying the pathlist of a directory on the command line causes the file to be restored in that directory. The directory structure and an index of the directory structure are created by fsave.

If more than one tape or disk is involved in the fsave backup, each tape or disk is considered to be a different volume. The volume count begins at one (1). When beginning a frestore operation, the last volume of the backup must be used first because it contains the index of the entire backup.

frestore first attempts to locate and read the index of the directory structure of the source device. frestore then begins an interactive session with the user to determine which file and directory in the backup should be restored to the current directory.

frestore Options

During the discussion of the actual frestore procedure, references are made to frestore options. The options are shown in Table 6-2.
### Table 6-2. frestore Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-?</td>
<td>Display the usage of frestore.</td>
</tr>
<tr>
<td>-b[=]&lt;int&gt;</td>
<td>Specify the buffer size used to restore the files.</td>
</tr>
<tr>
<td>-c</td>
<td>Checks the validity of files without using the interactive shell.</td>
</tr>
<tr>
<td>-d[=]&lt;path&gt;</td>
<td>Specify the source device. The default is /mt0.</td>
</tr>
<tr>
<td>-e</td>
<td>Display the pathlists of all files in the index, as the index is read from the source device.</td>
</tr>
<tr>
<td>-f[=]&lt;path&gt;</td>
<td>Restore from a file.</td>
</tr>
<tr>
<td>-i</td>
<td>Display the backup name, creation date, group.user number of the owner of the backup, volume number of the disk or tape and whether the index is on the volume. This option does not cause any files to be restored. The information is displayed, and frestore is terminated.</td>
</tr>
<tr>
<td>-p</td>
<td>Suppress the prompt for the first volume.</td>
</tr>
<tr>
<td>-q</td>
<td>Overwrite an already existing file when used with the -s option.</td>
</tr>
<tr>
<td>-s</td>
<td>Force frestore to restore all files from the source device without an interactive shell.</td>
</tr>
<tr>
<td>-t[=]&lt;dirpath&gt;</td>
<td>Specify an alternate location for the temporary index file.</td>
</tr>
<tr>
<td>-v</td>
<td>Display the same information as the -i option, but does not check for the index.                                                            This option does not cause any files to be restored. The information is displayed and frestore is terminated.</td>
</tr>
<tr>
<td>-x[=]&lt;int&gt;</td>
<td>Pre-extend the temporary file. &lt;int&gt; is given in kilobytes.</td>
</tr>
</tbody>
</table>
The Interactive Restore Process

Once `frestore` has been called, the following prompt is displayed:

```
frestore: mount the last volume
(press return when ready)
```

When you are ready, `frestore` reads the index and creates the directory structure of the backup. It then displays the prompt:

```
frestore>
```

This prompt indicates you are in the interactive shell. If the index is not on the mounted volume, `frestore` displays an error message and again prompts you to mount the last volume.

Once in the interactive shell, the `frestore` command and options are displayed when a return is typed at the prompt:

```
frestore> commands:
add [<path>] [-g=<group#> -u=<user#> -r -a] -- marks file for restoration
del [<path>] [-g=<group#> -u=<user#> -r -a] -- unmarks files for restoration
dir [<dir names>] [-e] -- displays a directory or directory
cht <path> -- changes directories within the restore file structure
pwd -- gives the pathlist to current dir in the restore file structure
cht <path> -- changes directories on target system rest [<path>] [-f -q] -- restores marked files in and below the current dir
check [-f] -- checks validity if marked files in and below the current dir
dump [<file>] -- dumps the contents of a file to stdout
list [<file>] -- list the contents of an ASCII file to stdout
$ -- forks a shell
quit -- quit frestore program
```

Options:

- `-g=<group#>` -- only mark files with 'group#'
- `-u=<user#>` -- only mark files with 'user#'
- `-r` -- mark directories recursively
- `-e` -- display directory with extended format
- `-f` -- force restoration of already restored files
- `-q` -- overwrite already existing files without question
- `-a` -- force marking or unmarking of an already restored file or dir
  
  * -- matches any string of characters on 'add' or 'del' only
  
  ? -- matches any single character on 'add' or 'del' only

```
frestore>
```
The index from the source device sets up a restore file structure paralleling the usual OS-9 file and directory structure.

The **dir** and **chd** shell command can display the restore file structure. For example:

```
frestore>dir
Directory of .
DIR1 file1 file2 file3
```

All files to be backed up onto the source device appear in the restore file structure regardless of what volume they appear in. Information concerning the file structure is available using the **-e** option with the **dir** command:

```
frestore>dir -e
Directory of .
Owner Last modified Attributes Volume Block Offset Size
Name ------ -------------- ----------- ------ ----- ------  -----   ---
1.23 89/08/22 16/14 ----r-wr 1 0 0 CF12 file1
1.23 89/08/25 11/00 ----r-wr 1 2 0 A356 file2
1.23 89/08/21 11/12 ----r-wr 1 4 0 45F0 file3
1.23 89/08/24 10/57 d-ewrewr 0 5 0 120 DIR1
```

In the interactive shell, you can mark the files you want restored with the **add** command. Groups of files can be marked using the **-g**, **-u** and **-r** options of the **add** command. The **-g** option marks files by group number. To mark files by user number, use the **-u** option. All directories within a specified directory may be marked by using the **-r** option.

- Files may be marked one at a time by specifying relative or complete pathlists within the restore file structure.
- An entire directory may be marked by specifying the pathlist of the directory.

Marking files does not restore them. It merely marks them as **to be restored**. You can see this when you use the **dir** command. Each file added to the **to be restored** list is marked by a plus sign (+) by its filename.
For example, the following directory has `file1` and `file2` marked for restoration, but `file3` is not marked. The directory `DIR1` and `DIR2` also have marked files:

```
frestore>add file1 file2 dir1/file5 dir1/file6 dir2/file7
frestore>dir
  Directory of .
  +DIR1  +DIR2  +file1  +file2
frestore>dir dir1
  Directory of DIR1
  file4  +file5  +file6
frestore>dir dir2
  Directory of DIR2
  +file7  file8
```

The `del` command can unmark files. Entire directories may be unmarked by specifying the directory name on the command line. If the `-r` option is also used, all files and directories included in the specified directory are unmarked. For example:

```
frestore>del -r dir2
frestore>dir
  Directory of . 10:42:32
  +DIR1  DIR2  +file1+file2
frestore>dir dir2
  Directory of DIR2
  file7  file8
```

Once files are marked, use the `rest` command to restore the current directory of the target device.

Files existing on the target system with the same name are overwritten without prompting if `del -q` is used. Otherwise, `frestore` displays the following prompt:

```
frestore: file1 already exists
  write over it or skip it (w/s)
```

An asterisk (*) preceding the name of a file in a `dir` listing indicates an error occurred while backing up this file. This file is incomplete and should not be restored.
The \texttt{cht} command allows you to change directories on the target device. This allows you to selectively restore files to specific directories.

After restoring files, you may continue marking files. Files previously restored have a hyphen (\texttt{-}) displayed next to their names in the restore file structure:

\begin{verbatim}
frestore>dir
     Directory of . 10:42:32
     -DIR1 DIR2 -file1 -file2
     file3
frestore>dir dir1
     Directory of DIR1
     file4 -file5 -file6
\end{verbatim}

There are two methods of restoring files more than once. The first method uses the \texttt{-a} option with the \texttt{add} command. This forces the file(s) previously marked as restored to be marked as “to be restored”. The second method requires the \texttt{-f} option to be used with the \texttt{rest} command. This forces any file previously marked as restored to be restored in the current directory.

The \texttt{-s} option forces \texttt{frestore} to restore all files and directories of the backup from the source device without the interactive shell.

Using the \texttt{-d} option allows you to specify a source device other than \texttt{/mt0}. For example, to restore all files/directories found on the source device \texttt{/mt1} to the directory \texttt{BACKUP} without using the interactive shell, type:

\begin{verbatim}
$ frestore -d=/mt1 -s BACKUP
\end{verbatim}

The \texttt{-v} option causes \texttt{frestore} to identify the name and volume number of the backup mounted on the source device. The date the backup was made and the group/user number of the person who made the backup are also displayed. This option does not restore any files. For example:

\begin{verbatim}
$ frestore -v
Backup: DOCUMENTATION
Made:  9/16/89 10:10
By:  0.0
Volume: 0
\end{verbatim}
The `-i` option displays the above information and also indicates whether the index is on the volume. Both the `-v` and `-i` options terminate `frestore` after displaying the appropriate information. These options are useful when trying to locate the last volume of the backup if any mix-up has occurred.

The `-e` option echoes each file pathlist as the index is read off the source device.

**Example Command Lines**

To restore files and directories from the source device `/mt0` to the current directory by way of an interactive shell, type the following command:

```
$ frestore
```

The following example restores files/directories from the source device `/d0` to the current directory using a 32-K buffer to write the restored files. As each file is read from the index, the file’s pathlist is echoed to the terminal.

```
$ frestore -eb=32 -d=/d0
```

**Incremental Backup Strategies**

Many different strategies are available for those concerned with regularly scheduled backups. Most strategies are well documented in computer books and magazines. The following two strategies are offered as examples.

**Daily Backup Strategy**

This strategy requires making a level 0 backup once every four weeks. Level 1, level 2, level 3, and level 4 backups are made on the weeks following the level 0 backup. Between each major backup, four daily backups are made: level 5, 6, 7, and 8. A recommended daily schedule is graphically presented in Figure 6-1.

This strategy is ideal for small microcomputer systems backed up by floppy disks. Mounting disks is much easier and faster than tapes. Each daily backup can usually be kept on one disk to make storage simple. This strategy is perfect for small timely backups with little redundancy in the backups.
One major disadvantage of this scheme is the restore time necessary in case of a major system failure such as a hard disk being formatted, erased or corrupted. Because of the lack of redundancy, more restore operations are necessary to re-create the systems file structure. On large systems with tape backups, this is a major consideration.

Figure 6-1. Day of Backup

Single Tape Backup Strategy

While most strategies rely on scheduled backup level changes, the single tape backup strategy depends on the size of the backup. The idea behind this strategy is to increase the level of the backup only when the backup cannot fit on a single tape. The only scheduled level backup is the level 0 backup. The level 0 backup occurs only when a higher level backup would not fit on a single tape or once a month, whichever occurs first. An example month’s schedule is graphically presented in Figure 6-2.
This strategy is suitable for tape backups of larger systems. Tapes are used efficiently because the question of how many tapes are needed never arises. This strategy also cuts down on person hours, tape mounting, and storage space used for tapes. It allows for enough redundancy to make restoring a full system relatively simple.

Disadvantages, however, do exist. Each time a backup is done, the size of the backup must be determined by using `fsave -s`. This takes an increasing amount of time, as the tape is filled.

**Use of Tapes or Disks**

Whatever strategy is used, you must make a decision concerning the number of tapes or disks to use. This decision must weigh the emphasis placed on redundancy, resources, person-hours, and storage. It must be offset with the possibility of tape or disk failure and system restoration.
In the first example strategy, the daily backups must be made on different volumes to overcome the lack of redundancy. The four daily volumes can be used week after week as daily backup volumes because of the lower level backups at the beginning of each week.

In the second example, theoretically, the same tape could be used for each day until a new level backup is reached. This ensures no redundancy and minimal storage. It is also the most risky in case of tape failure. Using a number of alternating tapes for each level down on storage allows a safety net in the case of tape failure. Using alternating level 0 tapes is another possibility.

**Using the tape Utility**

OS-9 provides a tape controller utility to facilitate setting up, reading and rewinding tapes from the terminal. When using tape media to backup or restore your system, the *tape* utility is very practical. The syntax of the *tape* utility is shown below:

```
tape {<opts>} [ <dev> ]
```

If the tape device `<dev>` is not specified on the command line and the `-z` option is not used, `tape` uses the default device `/mt0`.

*tape* has the following available options:

**Table 6-3. tape Options**

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-?</code></td>
<td>Display the use of tape.</td>
</tr>
<tr>
<td><code>-b[=&lt;num&gt;]&gt;</code></td>
<td>Skip a specified number of blocks. Default is 1 block. If <code>&lt;num&gt;</code> is negative, the tape skips backward.</td>
</tr>
<tr>
<td><code>-e=&lt;num&gt;</code></td>
<td>Erase a specified number of blocks of tape.</td>
</tr>
<tr>
<td><code>-f[=&lt;num&gt;]&gt;</code></td>
<td>Skip a specified number of tapemarks. Default is 1 tapemark. If <code>&lt;num&gt;</code> is negative, the tape skips backward.</td>
</tr>
<tr>
<td><code>-o</code></td>
<td>Put tape off-line.</td>
</tr>
<tr>
<td><code>-r</code></td>
<td>Rewind the tape.</td>
</tr>
<tr>
<td><code>-s</code></td>
<td>Determine the block size of the device.</td>
</tr>
<tr>
<td><code>-t</code></td>
<td>Retension the tape.</td>
</tr>
<tr>
<td><code>-w[=&lt;num&gt;]&gt;</code></td>
<td>Write a specified number of tapemarks. Default is 1 tapemark.</td>
</tr>
</tbody>
</table>
If more than one option is specified, `tape` executes each option function in a specific order. Therefore, it is possible to skip ahead a specified number of blocks, erase and then rewind the tape all with the same command. The order of option execution is as follows:

1. Get device name(s) from the `-z` option.
2. Skip the number of tapemarks specified by the `-f` option.
3. Skip the number of blocks specified by the `-b` option.
4. Write a specified number of tapemarks.
5. Erase a specified number of blocks of tape.

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
</table>
| `-z`    | Read a list of device names from standard input. The default is `/mt0`.
| `-z=<file>` | Read a list of device names from `<file>`.

Table 6-3. tape Options (Continued)
Chapter 6: Performing Backups

Backing Up the System Disk

Before you begin working with OS-9, make a backup of your master system disk. The backup procedure involves making an exact copy of a disk. It is important to have a backup copy of your system disk available in case your system disk becomes damaged; when damaged, the disk becomes unreadable.

Refer to the following documentation for more information:

- *Utilities Reference* (information about `format` and `backup`)
- *Chapter 4, The Shell* (a list of naming conventions used by OS-9)

Before you can back up your system disk, you need a properly formatted disk. OS-9 cannot read from or write to new disks until they have been formatted. The `format` utility initializes new disks for reading and writing. The OS-9 utility that makes copies of disks, `backup`, requires the backup disk to be the same size and format as the original disk.

The following section provides the steps you should take to back up a disk on a typical OS-9 system (booting from a floppy drive (usually called `/d0`)).

If you have a hard disk or are booting from a media other than a floppy disk, refer to *Chapter 7, OS-9 System Management*.

Before formatting your first disk, read the entire section on formatting disks.

The OS-9 system installation contains a menu-driven program, `install`, which optionally partitions and formats the destination drive and copies the OS-9 installation to that drive.

Formatting a Disk

The format of OS-9 system disks vary by the type of disk drive and by manufacturer. Usually, the format is set to the maximum capacity of the disk drive.

Refer to the *Basic File System Utilities* section in *Chapter 2, The OS-9 File System* for additional information about the format utility.
You can place several parameters on the command line with the `format` command, as shown in Table 6-4.

**Table 6-4. Command Line Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type of disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>-sd</td>
<td>single density</td>
</tr>
<tr>
<td>-dd</td>
<td>double density</td>
</tr>
<tr>
<td>-ss</td>
<td>single sided</td>
</tr>
<tr>
<td>-ds</td>
<td>double sided</td>
</tr>
</tbody>
</table>

Refer to your hardware documentation for the maximum capacity of your drives. Also, refer to the label of your system disk for the proper format of your backup copy.

**Multiple Drive Format**

If your system has two floppy disk drives, place the system disk in one drive and the new disk in the other drive. In multiple drive systems, one drive is normally labeled `/d0` and the other is labeled `/d1`. At the `$` prompt, type `format`, the drive name of the new disk, any desired options, then press the `<Return>` key to enter the command line. For example:

```
$ format /d1 -ds -dd
```

This command line asks the machine to format the disk in the second drive as a double-sided, double-density disk. You should adjust the options to conform to your disk format.

**Single Drive Format**

If your system has only one disk drive, you must load the `format` utility into memory.

Refer to the Basic File System Utilities section in Chapter 2, The OS-9 File System for more information about the load utility.

The `load` utility puts a copy of a program into the computer’s memory. To load the `format` utility into memory, type the following command at the `$` prompt:

```
load format
```
Once `format` has been loaded into memory, you can remove your system disk from the drive. OS-9 can execute the copy of `format` residing in memory. You can load and execute any OS-9 utility in this fashion.

Once you have loaded `format`, complete the following steps:

---

**Step 1.** Remove the system disk from the drive.

**Step 2.** Place the disk you are formatting into the drive.

**Step 3.** Enter the following at the `$` prompt to format the disk:

```
format /d0 -ss -dd
```

This command line specifies the disk should be formatted as a single-sided, double-density disk. Adjust the options as needed to conform to your disk format.

---

**Continuing the Formatting Process**

In the case of both single and multiple drive systems, `format` displays the specific disk format settings, followed by a prompt:

```
ready to format <drive name> (y/n/q)?
```

<drive name> is replaced by the name of the device you are trying to format, such as `/d0`.

⚠️ If the drive name in the prompt is not the name of the drive with the blank disk, type `q` to quit, or you may erase your only system disk.

Complete the following steps to continue the formatting process:

---

**Step 1.** If the drive name and parameters in the prompt are correct, type `y` for “yes”.

**Step 2.** If the values in the variable section are not correct, type `n` for “no”. `format` then prompts you for the changes to the current values of the options. After the variables have been set, you are queried again as to whether or not you want the disk to be physically formatted. This prompt is not issued for the `-np` option on the command line.
Step 3. If you type y at the prompt, you are asked for a name for the disk—unless you specified one with the -v option on the command line.

Step 4. You are then asked if you want to perform a physical verification. The physical verification process reads all sectors on the media and marks any bad sectors found as already allocated. This ensures the OS-9 file system does not attempt to use the bad sectors.

⚠️ Never back up a system disk to a disk having any bad sectors reported by format.

The Backup Procedure

After a disk is formatted, you can run `backup`. The `backup` utility makes an exact copy of the OS-9 system disk. There are other ways to make a copy of a disk, but this is the preferred method. The backup process involves copying the contents of your system disk to a formatted disk.

- During the backup procedure, the system disk is referred to as the source disk. The backup disk is called the destination disk.
- This procedure makes copies of any disk, not just the system disk.
- It is recommended that you write-protect your source disk when using the backup procedure. This prevents confusion when exchanging the source and destination disks.

`backup` makes two passes:

- The first pass reads a portion of the source disk into a buffer in memory and writes it to the destination disk.
- The second pass verifies everything was copied to the new disk correctly.

Generally, if an error occurs on the first pass, something is wrong with the source disk or the drive it is in.

If an error occurs during the second pass, the problem is with the destination disk. If `backup` repeatedly fails on the second pass, reformat the disk to make sure it has no bad sectors. If the disk reformats correctly, try the backup procedure again.
Multiple Drive Backup

If your system has two floppy disk drives, perform the following steps to make a multiple drive backup:

Step 1. Place the source disk in /d0.
Step 2. Place the destination disk in /d1.
Step 3. Type `backup` at the $ prompt.
Step 4. Press the <Return> key.

The system assumes you want to backup the disk in /d0. It responds to `backup` with the following prompt:

`ready to BACKUP /D0 to /D1?`

Step 5. Enter one of the following responses

- `y`: Type this key if the correct disks are in the correct drives.
- `q`: Type this key if the disks are not in the correct drives. Enter `q` to exit the backup process.

When you type `y`, the system copies all information on the disk in /d0 onto the disk in /d1 and returns the $ prompt.

Single Drive Backup

If your system has a single diskette drive, complete the following steps:

Step 1. Make sure your system disk is in /d0 and type the following command:

```
load backup
```

Step 2. Take your system disk out of /d0, and put your source disk in the disk drive (in this case, it is unnecessary as your system disk is your source disk). Type: `backup /d0`

This tells the system you are performing a single drive backup. The system responds with the following prompt:

`ready to BACKUP /D0 to /D0?`
Chapter 6: Performing Backups

Step 3. Enter one of the following responses

y Ready to perform the backup.
q Not ready to perform the backup. You exit the backup procedure when you enter q.

If you type y, the system begins a series of prompts to complete the backup procedure. This consists of swapping the source and destination disks in the disk drive as prompted by the system.

The first prompt is shown below:

ready destination, hit a key

Step 4. Remove the source disk from the drive.
Step 5. Insert the destination disk.
Step 6. Press any key to continue the backup procedure.

The next system prompt is shown below:

ready source, hit a key

Step 7. Remove the destination disk from the drive.
Step 8. Insert the source disk.
Step 9. Press any key to continue the backup procedure.
Step 10. Continue exchanging disks until the backup procedure is completed.

When you have backed up the system disk, store the original disk in a safe place and use the duplicate as your working system disk.

PC-AT system users must perform an additional step to back up the PC-AT system diskette. Refer to the OS-9 Porting Guide to find these steps.
System managers have a range of options to consider. OS-9 allows system managers to tailor their system to the needs of users.

This chapter discusses several topics with which system managers should become familiar. The following sections are included:

- Setting Up the System Defaults: the Init Module
- Extension Modules
- Changing System Modules
- Making Bootfiles
- Using the RAM Disk
- Making a Startup File
- Time Zones and the TZ Environment Variable
- System Shutdown Procedure
- Managing Processes in a Real-time Environment
- Using the tmode and xmode Utilities
- The termcap File Format
Setting Up the System Defaults: the Init Module

The Init module is sometimes referred to as the configuration module. It is a non-executable module located in memory in the sysboot file or in ROM. The Init module contains system parameters used to configure OS-9 during startup. The parameters set up the initial table sizes and system device names. For example, the amount of memory to allocate for internal tables, the name of the first program to run (usually either sysgo or shell), an initial directory, etc. are specified. You can examine the system limits defined in the Init module at any time.

The Init module must be present in the system in order for OS-9 to work.

The values in the Init module table are the system defaults. You can change these defaults by remaking the Init module. This is discussed later in this chapter.

The following is a list of the system defaults listed in the Init module. The fields in the Init module are defined by the structure init_data which is defined in init.h. The initialization macros are discussed later in this chapter.

Throughout this chapter, the system directory referred to are the defaults found in the Init module, unless otherwise specified.
The Init module system defaults are detailed in Table 7-1.

**Table 7-1. Init Module System Defaults**

<table>
<thead>
<tr>
<th>Name</th>
<th>Initialization Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_cachelist</td>
<td>CACHELIST</td>
<td>This is the offset to the cache region list declared in the default.des file in the port directory for the system.</td>
</tr>
<tr>
<td>m_compat</td>
<td>COMPAT</td>
<td>This byte is used for revision compatibility. The following bits are currently defined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bit 0: set to ignore sticky bit in module headers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bit 1: set to patternize memory when allocated and returned</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bit 2: set to inform the kernel not to automatically set the clock during coldstart</td>
</tr>
<tr>
<td>m_consol</td>
<td>CONS_NAME</td>
<td>This is the offset to the initial I/O pathlist string, usually /term.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This pathlist is opened as the standard I/O path for the initial process. It is generally used to set up the initial I/O paths to and from a terminal. This offset should contain zero if no console device is in use.</td>
</tr>
<tr>
<td>m_cpucompat</td>
<td>CPUCOMPAT</td>
<td>This field is reserved for system-specific flags.</td>
</tr>
<tr>
<td>m_cputyp</td>
<td>MPUCHIP</td>
<td>CPU type: 403, 603, 80386, etc.</td>
</tr>
<tr>
<td>m_dsptbl</td>
<td>DSPTBLSZ</td>
<td>This field contains the number of entries in the system call dispatch table. There must be at least 256 entries in this table, and each entry requires eight bytes.</td>
</tr>
</tbody>
</table>
Table 7-1. Init Module System Defaults (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Initialization Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_events</td>
<td>EVENTS</td>
<td>This is the initial number of entries allowed in the events table. If this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>table becomes full, it expands automatically. Refer to the OS-9 Technical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manual for specific information on events.</td>
</tr>
<tr>
<td>m_extens</td>
<td>EXTENSIONS</td>
<td>This is the offset to the name string of a list of customization modules, if</td>
</tr>
<tr>
<td></td>
<td></td>
<td>any.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A customization module is intended to complement or change existing standard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system calls used by OS-9. These modules are searched for at startup and are</td>
</tr>
<tr>
<td></td>
<td></td>
<td>usually found in the bootfile. If found, they are executed in system state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Module names in the name string are separated by spaces. The default name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string to be searched for is OS9P2. If there are no customization modules,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this value should be set zero.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE: Refer to the following section for more information on extension</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modules.</td>
</tr>
<tr>
<td>m_instal</td>
<td>INSTALNAME</td>
<td>This is the offset to the installation name string.</td>
</tr>
<tr>
<td>m_ioman</td>
<td>IOMAN_NAME</td>
<td>This is the offset to the name string of the module handling I/O system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls. This string is normally set to ioman.</td>
</tr>
<tr>
<td>m_maxage</td>
<td>MAXPTY</td>
<td>This is the initial system maximum natural age. m_maxage is discussed later</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in this chapter and in the OS-9 Technical Manual.</td>
</tr>
</tbody>
</table>
Table 7-1. Init Module System Defaults (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Initialization Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_memlist</td>
<td>MEMLIST</td>
<td>This is the offset to the memory list declared in default.des and defined in alloc.h. For a complete discussion on colored memory, see the OS-9 Technical Manual.</td>
</tr>
<tr>
<td>m_maxmem</td>
<td>MAXMEM</td>
<td>This field contains the top limit of free RAM.</td>
</tr>
<tr>
<td>m_maxsigs</td>
<td>MAXSIGS</td>
<td>This field specifies the default maximum number of signals queued up for a process.</td>
</tr>
<tr>
<td>m_minpty</td>
<td>MINPTY</td>
<td>This is the initial system minimum executable priority. m_minpty is discussed later in this chapter and in the OS-9 Technical Manual.</td>
</tr>
<tr>
<td>m_os9lvl</td>
<td>OS_LEVEL</td>
<td>OS-9 Level/Version/Revision/ Edition OS_VERSION OS_REVISION OS_EDITION. This four byte field is divided into three parts: level: 1 byte version: 2 bytes edition: 1 byte For example, level 2, version 2.0, edition 0 is 2200.</td>
</tr>
<tr>
<td>m_os9rev</td>
<td>OS_REVISION</td>
<td>This is the offset to the OS-9 level revision string.</td>
</tr>
<tr>
<td>m_paths</td>
<td>PATHS</td>
<td>This is the initial number of open paths in the system. If this table becomes full, it is expanded automatically.</td>
</tr>
<tr>
<td>Name</td>
<td>Initialization Macros</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>m_preio</td>
<td>PREIOS</td>
<td>This is an offset to the name string of a list of pre-I/O customization modules, if any.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These extension modules are initialized and called prior to the initialization of the I/O system during bootstrap. For more information on customization modules, refer to the description of m_extens and the following section.</td>
</tr>
<tr>
<td>m_procs</td>
<td>PROCS</td>
<td>This is the number of entries in the process descriptor table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this table becomes full, it is expanded automatically.</td>
</tr>
<tr>
<td>m_rtclock</td>
<td>RTC_NAME</td>
<td>This is the offset to the real-time clock module name string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The kernel attempts to call this module when the time is set, i.e. when _os_setime is called.</td>
</tr>
<tr>
<td>m_site</td>
<td>SITE</td>
<td>This field contains the installation site code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This user-definable field may be used to identify the site of the system.</td>
</tr>
<tr>
<td>m_slice</td>
<td>SLICE</td>
<td>This is the number of clock ticks per time-slice. The value is usually set to 1.</td>
</tr>
<tr>
<td>m_sparam</td>
<td>SYS_PARAMS</td>
<td>This is the offset to the parameter string (if any) to be passed to the first executable module.</td>
</tr>
</tbody>
</table>
Chapter 7: OS-9 System Management

Table 7-1. Init Module System Defaults (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Initialization Macros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_sysdrive</td>
<td>SYSDEVICE</td>
<td>This is the offset to the initial default directory name string, usually /d0 or /h0. The system initially does a chd and chx to this device prior to forking the initial device. If the system does not use disk, this offset must be zero.</td>
</tr>
<tr>
<td>m_sysgo</td>
<td>SYSSTART</td>
<td>This is the offset to the name string of the first executable module.</td>
</tr>
<tr>
<td>m_syspri</td>
<td>SYSPRIOR</td>
<td>This is the system priority at which the first module (usually Sysgo or Shell) is executed. This is generally the base priority at which all processes start. This value is commonly set to 128.</td>
</tr>
<tr>
<td>m_ticker</td>
<td>TICKNAME</td>
<td>This is the offset to the name string of the module used to generate the system clock tick. The kernel attempts to call this module when the first _os_setime system call is made.</td>
</tr>
<tr>
<td>m_ticksec</td>
<td>TICKSEC</td>
<td>This is the number of ticks a second of time is divided into. This value is usually set to 100.</td>
</tr>
<tr>
<td>m_tmzone</td>
<td>SYS_TLMZONE</td>
<td>This is the system time zone in minutes offset from Greenwich Mean Time (GMT). This field would be 360 for a system six time zones west of GMT and -360 for a system six time zones east of GMT.</td>
</tr>
<tr>
<td>m_usract</td>
<td>USRACCT</td>
<td>This is the offset to the name string of the user accounting module.</td>
</tr>
</tbody>
</table>

For more information on the Init module, see the OS-9 Technical Manual.
Extension Modules

Extension modules can be attached to OS-9 during the system cold-start procedure to increase the functionality of OS-9. Extension modules can be used for a variety of functions such as user accounting, system security, and system caching.

In the Init module, the m_extens offset points to a list of module names. By default, the name of the list is OS9P2. If the modules are found during cold-start, they are called. If an error is returned, the system stops. Three of these modules are listed below:

- **Cache**
  - The cache module enables the system to control any hardware caches present.
  - This module can be customized to take advantage of any cache hardware the system may have.

- **SSM**
  - The system security module (SSM) enables memory protection.

- **FPU**
  - The floating point unit (FPU) module currently supplies five functions.
    - These functions include saving, loading, and resetting the floating point processor content; setting a null context for a process; and testing for a null context.

Also, in the Init module, the m_preio offset points to a list of module names that are initialized during bootstrap prior to the initialization of the I/O system. This enables the installation of services that may be required during the initialization of the I/O system.

Changing System Modules

The provided system modules have been configured to satisfy the needs of the majority of users. However, you may wish to alter the existing modules or create new modules. New system modules and alterations to existing system modules can be made by changing the defaults in the systype.h file. The system modules most commonly altered are the device descriptors and the Init module.
The `systype.h` file is located in the `PORTS` directory. It contains macros such as `TERM`, `DiskH0`, and others for each device descriptor and the `Init` module. These macros contain basic memory map information, exception vector methods (for example, vectors in RAM or ROM), I/O device controller memory addresses and initialization data, and other information for each device descriptor and the `Init` module.

The `systype.h` file consists of five main sections used when installing OS-9:

- **Init module** `CONFIG` macro
- SCF Device Descriptor macros and definitions
- RBF Device Descriptor macros and definitions
- ROM configuration values
- target system specific definitions

The macros related to the `Init` module are surrounded in `systype.h` with `#if defined(INITMOD)`. The definitions provided here override the default values when the `Init` module is made. This allows port-specific system tuning without modifying the generic file that all ports use to define the system configuration.

The macros device descriptors are surrounded in `systype.h` with `#if defined(<desc>)` where `<desc>` is the name of the descriptor being created. For example, you’ll find a pre-processor directive like `#if defined(TERM)`. The macros following this line, up to the corresponding `#endif`, relate to the `TERM` macro for your machine. The fields affected by these macros are discussed more fully in the OS-9 Technical Manual.

The ROM configuration values appear in `systype.h` surrounded by `#if defined(CNFGDATA)`. These definitions control how your ROM modules behave for your particular port. These definitions and their effects are discussed more fully in the BSP Reference or OS-9 Porting Guide provided with your package.

System specific definitions, such as control register and vectors, should be placed in `systype.h`. This allows the system-specific definitions to be maintained in a single, system-specific file.

For more information on the make utility, refer to the chapter on making files and the make utility description in the Utilities Reference manual.
To change your system configuration, change the definitions appearing in your port `systype.h` file with any text editor. Since all relevant system components include `systype.h`, the change takes place the next time they are regenerated.

Use the `make` utility to regenerate the appropriate system components. Running the makefile in your `PORTS` directory regenerates all the port specific modules for your system. Since your changes likely only affect a small subset of these modules, you should find the makefile that is relevant to the changes you have made. For example, to change the baud rate of the `/t1` device, find the makefile for that descriptor (`SCF/SC16550/DESC`) and execute it. This regenerates the `/t1` device descriptor.

**Making Bootfiles**

A bootfile contains a list of modules to be loaded into memory during the system’s bootstrap sequence. The provided bootfiles have been configured to satisfy the majority of users, but you may want to add or remove modules from an existing bootfile.

**Bootlist Files**

Bootfiles are usually created using a `bootlist` file and the `-z` option of `bootgen` utility. The `bootlist` file contain a list of files, one file per line, to use in creating the bootfile. Using a `bootlist` file is a convenient way to maintain bootfile contents, as the `bootlist` file can easily be edited. The `bootlist` files are usually located in the `ports` directory (for example: `/h0/MWOS/OS9000/603/PORTS/MVME1603`) along with the individual files used for constructing the bootfile.

**Bootfile Requirements**

The contents and module order of a bootfile are usually determined by the end-user’s system configuration and requirements. However, the following points should be noted when you construct a bootfile:

- The kernel must be present in the system, either in ROM or in the bootfile. If the kernel is in the bootfile, it must be the first module.
- The `Init` module must be present in the system, either in ROM or in the bootfile.

All other modules are dependent upon the system configuration.
Making RBF Bootfile

To make a bootfile for an RBF device (hard disk or floppy disk), you need to edit the bootlist file to match your requirements and then run the bootgen utility:

```
chd /h0/MWOS/OS9000/<CPU-family>/PORTS/<processor>/BOOTLIST
<edit bootlist file>
bootgen <device> -z=<bootlist>
```

For example:

```
chd /h0/MWOS/OS9000/80386/PORTSPCAT/BOOTLIST
```

The `<device>` specified is the disk on which you want to install the bootfile. If this device is a hard disk, specify the format-enabled device name (for example, h0fmt).

For example, to make a floppy-disk bootfile, type the following command: `bootgen /d0 -z=d0_765.bl`

To make a hard disk bootfile, type the following command:

`bootgen /h0fmt -z=h0_ide.bl`

Using the RAM Disk

OS-9 provides support for RAM disks. These disks reside solely in Random Access Memory (RAM). The information stored on a RAM disk can be accessed significantly faster than the same information stored on a hard or floppy disk. Any file may be stored and accessed on a RAM disk. To use a RAM disk, you must have a device descriptor, a RAM disk driver and the RBF file manager.

In many system configurations, a RAM disk is used as the default system device. When the RAM disk is used as the default system device, it is known as device `dd`, instead of `r0`. The name of the device descriptor is `.r0.dd`. Using this descriptor allows compilers to use the RAM disk as a “fast access” device for temporary file. The RAM disk is usually initialized at startup with definition and library files, if it is to be used as the default system device.
RAM disks are either volatile or non-volatile. A volatile RAM disk disappears when the system is reset or the power is shut off. A non-volatile RAM disk resides in a place such as battery backed up RAM and does not disappear when the system is reset or powered down.

**Volatile RAM Disks**

Volatile RAM disks may be allocated memory either from free system memory or from outside free system memory. Volatile RAM disks not allocated from the free system memory must not be part of the system memory list, and they must have a port address greater than or equal to 1024. This port address indicates the actual start address of the RAM disk.

**Non-Volatile RAM Disks**

A non-volatile RAM disk must be located in an area of memory the system will not try to allocate. If it is located in an area known to the system, the RAM disk may be cleared because the memory is assumed to be un-allocated and may later be used by the system. In addition, the format protect bit must be set for non-volatile RAM disks and the port address must be greater than or equal to 1024.

**Making a Startup File**

Using bootfiles is not the only way of loading modules and device into memory at the time of startup. A startup procedure is executed each time OS-9 is booted and the standard `sysgo` is used. On disk-based systems, the startup procedure executes a `startup` file. The `startup` file is located in the `sys` directory in the root directory of the system disk.

The `startup` file is an OS-9 procedure file. It contains OS-9 commands to be executed immediately after booting the system.

While some modules and devices, such as the kernel, should be loaded from the `sysboot` file, having the `startup` file load most modules can be advantageous. For example, it is easier to upgrade a system by modifying the `startup` file. To change this file, you simply use a text editor and make the changes. To change the `sysboot` file, you must also use the `bootgen` utility.
A procedure file is made up of executable commands. Each command is executed exactly as if it were entered from the shell command line. Each line starting with an asterisk (*) is a comment and is not executed.

From the root directory, the startup file can be examined by entering the following command:

```
$ list sys/startup
```

A listing similar to the following is displayed:

```
-t -np
*
* OS-9000
* Copyright 1996 by Microware Systems Corporation
* Copyright 2001 by RadiSys Corporation
*
* The command in this file are highly system dependent and should
* be modified by the user.
*
* setime; * start system clock
link shell csl ; * make "shell" and "csl" stay in memory
* iniz r0 h0 d0 t1 p1 ; * initialize device
* load -z=sys/loadfile ; * make some utilities stay in memory
* load bootobjs/r0.dd ; * get default device descriptor
* tsmon /t1 & ; * start other terminals
list sys/motd
```

The first executable line, `-t -np`, turns on the talk mode option of the shell and turns off the prompt option for the duration of this procedure. The talk mode option echoes each executed command to the terminal display. This allows you to see what command are being executed.

For systems with battery-backed up clocks, run `setime` with the `-s` option to start time-slicing. The date and time are read from the clock.

The other executable lines in the distributed startup file are followed by a comment explaining the purpose of the command. Some standard commands are provided as comments. If you want the command executed during the startup procedure, use a text editor to remove the asterisk preceding the command.

For example, to execute the `setime` command when the startup file is executed, remove the asterisk preceding the command.
If you are concerned that your system may be unusable due to a corrupt or missing startup file, set the parameter string in the init module as shown below:

```
...;mshell --lp=backup- &; profile /h0/sys/startup & 
```

This setting first brings up mshell, then profiles the startup files. (This may result in the forking off of more than one mshell.) Once you are satisfied the startup file is present and functioning, you can change your init string back to the following setting:

```
...;/h0/sys/startup & 
```

**Initializing Devices: iniz r0 h0 d0 t1 p1**

The `iniz r0 h0 d0 t1 p1` commented command initializes the following specific devices:

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>RAM disk</td>
</tr>
<tr>
<td>h0</td>
<td>Hard disk</td>
</tr>
<tr>
<td>d0</td>
<td>Floppy disk</td>
</tr>
<tr>
<td>t1</td>
<td>Terminal</td>
</tr>
<tr>
<td>p1</td>
<td>Serial Printer</td>
</tr>
</tbody>
</table>

When OS-9 opens a path to a device, it first checks to see if the device is known to OS-9. To be known, a device must be initialized and memory must be allocated for its device driver. If the device is unknown at the time of the request, OS-9 initializes the device, allocates memory and opens the path. For example, a simple `dir /d0` command initiates this sequence of events if `d0` has not been previously initialized.

The `iniz` utility initializes devices. `iniz` performs an `I_ATTACH` system call on each device name passed to it. This initializes and links the device to the system.

To initialize a device after the system has been started, type `iniz` and the name(s) of the device(s) to attach to the system. `iniz` goes through the procedure of initializing the device(s) and allocating the memory needed for the device. If the device is already attached, it is not re-initialized, but the link count is incremented.
For example, to increment the link count of modules, t2 and t3, type the following command:

```
$ iniz t2 t3
```

The device names can be read from standard input with the `-z` option or from a file with the `-z=<file>` option. To increment the link counts of devices listed in a file called `/h0/add.file`, type the following command:

```
iniz -z=/h0/add.files
```

You can use the `deiniz` utility to deinitialize a device. `deiniz` checks the link count before removing the device from storage. If the link count is greater than one, `deiniz` lowers the link count. If the link count is one, `deiniz` lowers the link count to zero, and removes the device from the system device table. The device then becomes unknown to OS-9.

Non-sharable devices must be placed in a bootfile to become known to the system. If a non-sharable device is initialized, it is unusable because the link count has been incremented, which makes it appear to be in use.

To use the `deiniz` utility, type `deiniz` followed by the name(s) of the devices(s) to remove from the system.

For example, to decrement the link count of module p2, type the following command:

```
$ deiniz p2
```

deiniz can read the device names from standard input with the `-z` option or from a file with the `-z=<file>` option. To remove the file listed in a file called `/h0/not.needed`, type the following command:

```
$ deiniz -z=/h0/not.needed
```

This initialize/de-initialize sequence can result in slower execution of programs and may cause memory fragmentation problems. To avoid these symptoms, it is recommended that all devices connected to the system at startup be initialized in the startup file.

For more information on the `iniz` and `deiniz` utilities, refer to the Utilities Reference manual.
Initializing the connected device at startup initializes the device and allocates memory for its driver for the duration of the time the system is running, unless specifically de-initialized. For example, a system with two floppy drives and one hard disk drive can initialize these devices in the startup file:

```
iniz h0 d0 d1 t1 p1 p
```

**Loading Utilities Into Memory: load -z=sys/loadfile**

The next line of the startup file loads a number of utilities into memory. If a utility is not already in memory, it must be loaded into memory before it is used. Pre-loading basic utilities at startup time avoids the necessity of loading the utility each time it is executed.

To load utilities into memory at startup, you must create a file containing the names of each utility to load, one utility per line. While the file may have any name, Microware recommends `loadfile`. You can locate this file in any directory as long as its location is specified on the command line. If `loadfile` is located in the `SYS` directory, the startup file command line will look like the following example:

```
load -z=sys/loadfile
```

Previous versions of the operating system had the following commented line in the startup file:

```
load utils
```

This method involved creating a `utils` file by merging the desired utilities into a single file in the command directory. While you may still use this method, using `loadfile` is preferable because it uses less disk space and is easier to edit.
Loading the Default Device Descriptor: Load `bootobjs/r0.dd`

Many OS-9 compilers and application programs look for definition files and libraries in directories located on the default system device. The default system device is known as `dd`. `dd` may be defined as any disk device, but it is usually synonymous for one of the following devices:

Table 7-3. Disk Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>RAM disk</td>
</tr>
<tr>
<td>h0</td>
<td>Hard disk</td>
</tr>
<tr>
<td>d0</td>
<td>Floppy disk</td>
</tr>
</tbody>
</table>

If a default device is to be used (`dd`) and the device descriptor is not in the bootfile, then the device descriptor must be loaded. The next line in the startup file loads the device descriptor. The default device used is the RAM disk named `r0`. If you want another device to be the default device descriptor, change the `.r0` extension to reflect the appropriate device. If you have a `dd` device in your bootfile or if no default device is to be used, leave this line as a comment.

Multi-user Systems: `tsmon /t1` &

The `tsmon` utility is used to make your system a multi-user system. This utility supervises idle terminals and initiates the login procedure for multi-user systems. The startup file command line, `tsmon /t1&`, initiates the time-sharing monitor on the serial port `/t1`.

`tsmon` can monitor up to 28 device name pathlists. Therefore, if you have multiple devices for `tsmon` to monitor, you can name up to 28 devices on each `tsmon` command line. Use the `ex` built-in shell command to execute `tsmon` without creating another shell. This conserves system memory. For example:

```
ex tsmon /term /t1 /t2 /t3 /t4 /t5&
```

When a carriage return is entered on any of the specified paths, `tsmon` automatically forks `login` and standard I/O paths are opened to the device.

For more information on the `tsmon` utility, refer to the Utilities Reference manual.
The login procedure uses the password file located in the SYS directory for individual login validation. The provided password file has two example login entries. Each of the fields in an entry in the password file is explained in the chapter on the shell and in the login utility description in the Utilities Reference manual. If login fails because you could not supply a valid user name or password, control returns to tsmon.

**Time Zones and the TZ Environment Variable**

The TZ environment variable is used to specify the time zone for the C functions to use. TZ should have the following format:

```
zzz[+/-n][:ddd]
```

In addition, the value of m_tmzone in the init module should be set to match the system time zone offset from GMT when daylight savings time is not in effect.

You must set the TZ environment variable correctly in order to accurately track daylight savings time. However, if it is not necessary to track daylight savings time, you do not need to set TZ—provided that m_tmzone in the init module has been set correctly.

Refer to the Ultra C Library Reference for more information about the TZ environment variable and time functions.

**Time Zones**

The following table gives brief descriptions of each time zone.

<table>
<thead>
<tr>
<th>Zone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMT/UTC</td>
<td>Greenwich Mean Time (or Coordinated Universal Time)</td>
</tr>
<tr>
<td>PST/PDT</td>
<td>USA Pacific Standard Time/Daylight Savings Time</td>
</tr>
<tr>
<td>MST/MDT</td>
<td>USA Mountain Standard Time/Daylight Savings Time</td>
</tr>
<tr>
<td>CST/CDT</td>
<td>USA Central Standard Time/Daylight Savings Time</td>
</tr>
<tr>
<td>EST/EDT</td>
<td>USA Eastern Standard Time/Daylight Savings Time</td>
</tr>
<tr>
<td>YST</td>
<td>Yukon Standard Time (Most of Alaska)</td>
</tr>
<tr>
<td>AST</td>
<td>Aleutian/Hawaiian Standard Time</td>
</tr>
<tr>
<td>EET</td>
<td>Eastern European Time</td>
</tr>
</tbody>
</table>
System Shutdown Procedure

There are times when you want to shut your system down. When you reset or power down your system, you may need to do more than just press the reset button. Certain programs need to be shut down gracefully. For example, most network communications, print spools, and inter-system processes need special attention. These processes may have options or other arrangements needing consideration before shutting down your system.

<table>
<thead>
<tr>
<th>Zone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CET</td>
<td>Central European Time</td>
</tr>
<tr>
<td>WET</td>
<td>Western European Time</td>
</tr>
<tr>
<td>JST</td>
<td>Japan Standard Time</td>
</tr>
<tr>
<td>MIT</td>
<td>Midway Islands Time</td>
</tr>
<tr>
<td>HST</td>
<td>Hawaii Standard Time</td>
</tr>
<tr>
<td>PNT</td>
<td>Phoenix Standard Time</td>
</tr>
<tr>
<td>IET</td>
<td>Indiana Eastern Standard Time</td>
</tr>
<tr>
<td>PRT</td>
<td>Puerto Rico Standard Time</td>
</tr>
<tr>
<td>CNT</td>
<td>Canada Newfoundland Time</td>
</tr>
<tr>
<td>AGT</td>
<td>Argentina Standard Time</td>
</tr>
<tr>
<td>BEZ</td>
<td>Brazilillian Standard Time</td>
</tr>
<tr>
<td>CAT</td>
<td>Central Africa</td>
</tr>
<tr>
<td>ECT</td>
<td>European Central Time</td>
</tr>
<tr>
<td>ART</td>
<td>Arabic Standard Time</td>
</tr>
<tr>
<td>EAT</td>
<td>Eastern African Time</td>
</tr>
<tr>
<td>MET</td>
<td>Middle Eastern Time</td>
</tr>
<tr>
<td>NET</td>
<td>Near East Time</td>
</tr>
<tr>
<td>PLT</td>
<td>Pakistan Lahore Time</td>
</tr>
<tr>
<td>IST</td>
<td>India Standard Time</td>
</tr>
<tr>
<td>BST</td>
<td>Bangladesh Standard Time</td>
</tr>
<tr>
<td>VST</td>
<td>Vietnam Standard Time</td>
</tr>
<tr>
<td>CTT</td>
<td>China Taiwan Standard Time</td>
</tr>
<tr>
<td>ACT</td>
<td>Australia Central Time</td>
</tr>
<tr>
<td>AET</td>
<td>Australian Eastern Time</td>
</tr>
<tr>
<td>SST</td>
<td>Solomon Standard Time</td>
</tr>
<tr>
<td>NST</td>
<td>New Zealand Standard Time</td>
</tr>
</tbody>
</table>
In addition to taking care of processes requiring special attention, you should prepare the system users for the shutdown. If at all possible, users should be allowed enough time to save their file and close their workstation. One way of alerting users that the system is going down is by echoing a message using the `echo` and `tee` utilities. However, you should realize messages sent over the system in this manner are not seen by users who do not press a carriage return after the message has been sent. For example, if a programmer is sitting at a shell prompt, the message does not appear on the terminal screen until a carriage return is entered.

In this case, verbal warnings are important. This means in addition to sending a warning message out over the system, you may want to use either an intercom system or the telephone to talk to each person connected to the system.

You can simplify the process of actually shutting down your system by creating a procedure file. Once created, you can run the procedure from the shell command line prompt or a separate password entry may be created for the sole purpose of shutting down the system.

For example, if you have a procedure file called `shutdown.sys`, you could create the following password file entry:

```
sys,shutdown,0.0,128,,sys,shell shutdown.sys
```

Once you login as user `sys` with password `shutdown`, the shutdown procedure begins because the system immediately has the shell execute the `shutdown.sys` file. The following code is an example of a useful procedure file for shutting down the system:

```
-t -nx -np
*  
* System Shutdown Procedure  
*  
  echo WARNING The system will shut down in 3 minutes ! tee /t1 /t2 /t3 /t4 /t5  
  sleep -s 60  
  echo WARNING The system will shut down in 2 minutes ! tee /t1 /t2 /t3 /t4 /t5  
  sleep -s 115  
  echo WARNING 5 seconds to system shut down ! tee /t1 /t2 /t3 /t4 /t5  
  sleep -s 5  
  spl -$; * terminate spooler  
  sleep -s 3; * wait 3 seconds  
  break; * call ROM debugger
```
The first six commands after the comment identifying the procedure function broadcast three warnings to the terminals on the system. The first warning tells the users the system is going down. The other two warnings serve as reminders.

The remaining command lines shut down the system:

Table 7-5. Command Lines

<table>
<thead>
<tr>
<th>Command line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spl -$</td>
<td>This command terminates the spooler. All unfinished jobs are lost when the spooler is terminated.</td>
</tr>
<tr>
<td>sleep -s 3</td>
<td>This command causes the system to wait three seconds before executing the next command line. This allows the previous command time to complete execution.</td>
</tr>
<tr>
<td>break</td>
<td>This command sends a break call to the low-level debugger. When this debugger receives this call, it takes control of the system.</td>
</tr>
</tbody>
</table>

Managing Processes in a Real-time Environment

The ability to manage processes in a real-time environment is one of the advantages of OS-9. OS-9 has three primary methods by which system managers can manage processes in a real-time environment:

- Manipulating process priority.
- Using `d_minpty` and `d_maxage` to alter the system process scheduling.
- Having system-state processes as well as user-state process.

Manipulating the Priority of a Process

When processes are executed on the command line, their initial priorities can be changed using the process priority modifiers discussed in the chapter on the shell. This enables users with a crucial task to set the priority on their process higher so it runs sooner and more often than less crucial processes.

The initial priority is also a parameter for the fork and chain system calls.
Changing the System’s Process Scheduling

The way OS-9 schedules processes can be affected by the \texttt{d\_minpty} and \texttt{d\_maxage} system global variables. \texttt{d\_minpty} and \texttt{d\_maxage} are available to super users through the \texttt{\_os\_setsys} system call. These system variables can be used to effect the aging of processes.

The initial priority of a process is aged each time it is passed by for execution while it is waiting for CPU time.

\texttt{d\_minpty} defines a minimum priority below which processes are neither aged nor considered candidates for execution. Processes with priorities less than \texttt{d\_minpty} remain in the active queue and continue to hold any system resources they held before \texttt{d\_minpty} was set.

\texttt{d\_minpty} is usually set to zero. All processes are eligible for aging and execution when this value is set to zero because all processes have an initial priority greater than zero.

If you have a critical process needing to be run and several other users have processes they want to run, use the process priority modifier to increase the priority of the critical process. Then, set \texttt{d\_minpty} to a value less than the priority you assigned to the critical process but greater than the priority of the other processes. The critical process now continues using the CPU until another process with a priority greater than \texttt{d\_minpty} is entered into the active queue or the critical process is finished.

For example, if \texttt{d\_minpty} is set to 500 and you set the priority of your process at 600, your process continues to use the CPU while processes with priorities less than 500 are not able to run until \texttt{d\_minpty} is reset.

\texttt{d\_minpty} is potentially dangerous. If the minimum system priority is set above the priority of all running tasks, the system completely shuts down and can only be recovered by a reset. It is crucial to restore \texttt{d\_minpty} to zero when the critical task finishes or to reset \texttt{d\_minpty} or a process’ priority in an interrupt service routine.

\texttt{d\_maxage} defines a maximum age over which processes are not allowed to mature. By default, this value is set to zero. When \texttt{d\_maxage} is set to zero, it has no effect on the processes waiting to use the CPU.
When set, \( d_{\text{maxage}} \) essentially divides tasks into two classes: low priority and high priority. A low priority task is considered to be any task with a priority below \( d_{\text{maxage}} \). Low priority tasks continue aging until they reach the \( d_{\text{maxage}} \) cutoff, but they are not executed unless there are no high priority tasks waiting to use the CPU.

A high priority task is any task with a priority above \( d_{\text{maxage}} \). A high priority task receives the entire available CPU time, but it is not aged. When the high priority task(s) are inactive, the low priority tasks run.

For example, if \( d_{\text{maxage}} \) is set to 2000 and three processes with initial priorities of 128 are in the active queue, the processes run just as if \( d_{\text{maxage}} \) had not been set. Then, if a process with an initial priority of 2500 is entered into the active queue, it receives CPU time when the process currently in the CPU has finished. Once using the CPU, the high priority process runs uninterrupted until a process with a higher priority is entered into the active queue or the process finishes. When the process finishes executing, the low priority processes again are able to use the CPU.

Any process performing a system call is not preempted until the call is finished, unless the process voluntarily gives up its timeslice. This exception is made because these processes may be executing critical routines affecting shared system resources and could be blocking other unrelated processes.

**Using System-State Processes and User-State Processes**

The second method OS-9 uses to manage real-time priority processing is the existence of system-state processes. System-state processes are processes running in a supervisor or protected mode. System-state processes basically have unlimited access to system memory and other resources. When a process in system state wants to use the CPU, it waits until it has the highest age.

User-state processes do not have access to all points in memory and do not have access to all of the commands. When a process in user-state gains time in the CPU, it runs only for the time specified by the timeslice. When it has finished using its timeslice, it is entered back into the active queue according to its initial priority.
Using the tmode and xmode Utilities

The `tmode` and `xmode` utilities are also available to help you customize OS-9. Use the `tmode` utility to display or change the operating parameters of the user’s terminal. The `xmode` utility is similar to the `tmode` utility. Use the `xmode` utility to display or change the initialization parameters of any SCF-type device such as a video display, printer, or RS-232 port. Some common uses are to change the baud rates and control key definitions.

Using the tmode Utility

To use the `tmode` utility, type `tmode` and any parameter(s) you need changed. If no parameters are given, the present values for each parameter are displayed. Otherwise, the parameter(s) given on the command line are processed. You can pass any number of parameters on a command line. Each parameter is separated by a space.

If a parameter is set to zero, OS-9 no longer uses the parameter until it is re-set to a code OS-9 recognizes. For example, the following command sets the `<tab>` and `<bell>` output characters to zero.

```
tmode tab=0x00 bell=0x00
```

OS-9 does not output tabs or bells until the values are re-set.

The `tmode` parameters are documented in the Utilities Reference manual.

To re-set the values of a parameter to their default as given in this manual, specify the parameter with no value.

You can use the `-w=<path#>` option to specify the path number to be affected. If a path number is not provided, standard input is affected.

If `tmode` is used in a shell procedure file, the option `-w=<path#>` must be used to specify one of the standard paths (1 or 2) to change the terminal’s operating characteristics. The change remains in effect until the path is closed.

To effect a permanent change to a device characteristic, you must first initialize the device, and then use the `xmode` utility to alter the device’s initial operating parameters.
Using the xmode Utility

To use the xmode utility, type xmode and any parameter(s) to change. If no parameters are given, the present values for each parameter are displayed. Otherwise the parameter(s) given on the command line are processed. You can give any number of parameters on a command line. Each parameter is separated by spaces or commas. You must specify a device name if the given parameter(s) are to be processed.

The xmode parameters are documented in the Utilities Reference manual.

Like tmode, if a parameter is set to zero, the device no longer uses the parameter until it is re-set to a recognizable code. To re-set the values of parameters to their default, specify the parameter with no value. This re-sets the parameter to the default value as given in this manual.

Using xmode, you can also define control keys affecting the input line. For example, <control>B is, by default, defined as a backspace key for the command line. You can use xmode to redefine <control>B to perform another function or to redefine another key to backspace on the input line.

The termcap File Format

The termcap file is a text file containing control code definitions for one or more types of terminals. Each entry is a complete description list for a particular kind of terminal.

The first section of a termcap entry is divided into three parts.

- a two character entry
- the most common name
- a long name

Each part is a different way of naming the terminal. A bar (|) character separates the parts of a termcap entry. The first part is a two character entry. The second part is the most common name for the terminal. This name must contain no blanks. The final part is a long name fully describing the terminal. This name may contain blanks for readability. For example:

kh|abm85h|kimtron abm85h:
You can check the values stored in `TERM` by using the `printenv` command:

```
$ printenv
TERM=abm85h
```

You must set the `TERM` environment variable to the name used in the second part of the name section. In the following example, `TERM` is set to `abm85h`:

```
$ setenv TERM abm85h
```

The rest of the entry consists of a sequence of control code specifications for each control function. Each item in the list is separated by a colon (`:`) character. An entry may be continued onto the next line by using a backslash (`\`) character as the last character of the line. It must appear after the last colon of the previous item. The next line must begin with a colon. For example:

```
ka|amb85|kimtron abm85:\
 :ct=\E3: ... 
```

Each item begins with a terminal capability. Each capability is a two character abbreviation. Each capability is either a boolean itself or it is followed by a string or a number. If a boolean capability is present in the `termcap` entry, then the capability exists on that terminal.

All numeric capabilities are followed by a pound sign (`#`) and a number. For example, the number of columns capability for an 80 column terminal could be described as follows:

```
co#80:
```

All string capabilities are followed by an equal sign (`=`) and a character string. A time delay in milliseconds may be entered directly after the equal sign (`=`) if padding is allowed in that capability. The padding characters are supplied by `tputs()` after the remainder of the string is transmitted to provide the time delay. The time delay may be either an integer or a real. The time delay may be followed by an asterisk (`*`). The asterisk specifies the padding is proportional to the number of lines affected.
It is often useful to specify the time delay using the real format. For example, the clear screen capability is specified as ^z with a time delay of 3.5 milliseconds by the following entry:

```c
cl=3.5*^z:
```

Escape sequences are indicated by a \E. A control character is indicated by a circumflex (^) preceding the character. The following special character constants are supported:

### Table 7-6. Supported Special Character Constants

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Character</th>
<th>Hexadecimal code</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>backspace</td>
<td>($08)</td>
</tr>
<tr>
<td>\f</td>
<td>formfeed</td>
<td>($0C)</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
<td>($0A)</td>
</tr>
<tr>
<td>\r</td>
<td>return</td>
<td>($0D)</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
<td>($09)</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
<td>($5C)</td>
</tr>
<tr>
<td>^</td>
<td>circumflex</td>
<td>($5E)</td>
</tr>
</tbody>
</table>

Characters are specified as three octal digits after a backslash (\). For example, if a colon must be used in a capability definition, it must be specified by \072. If it is necessary to place a null character in a capability definition use \200. C routines using termcap strip the high bits of the output, therefore \200 is interpreted as \000.

### termcap Capabilities

The following table contains a list of termcap capabilities recognized by termcap. Not all of these capabilities need to be present for most programs to use termcap. They are provided for completeness. (P) indicates padding may optionally be specified. (P*) indicates the optional padding may be based on the number of lines affected:

### Table 7-7. termcap Capabilities

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Padding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ae</td>
<td>string</td>
<td>(P)</td>
<td>End alternate character set.</td>
</tr>
<tr>
<td>al</td>
<td>string</td>
<td>(P*)</td>
<td>Add new blank line.</td>
</tr>
<tr>
<td>am</td>
<td>boolean</td>
<td></td>
<td>End alternate character set.</td>
</tr>
<tr>
<td>as</td>
<td>string</td>
<td>(P)</td>
<td>Start alternate character set.</td>
</tr>
<tr>
<td>bc</td>
<td>string</td>
<td></td>
<td>Backspace if not ^H.</td>
</tr>
</tbody>
</table>
### Table 7-7. termcap Capabilities (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Padding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>boolean</td>
<td></td>
<td>Terminal can backspace with (^H).</td>
</tr>
<tr>
<td>bt</td>
<td>string</td>
<td>(P)</td>
<td>Back tab.</td>
</tr>
<tr>
<td>bw</td>
<td>boolean</td>
<td></td>
<td>Backspace wraps from column 0 to last column.</td>
</tr>
<tr>
<td>cc</td>
<td>string</td>
<td></td>
<td>Command character in prototype if terminal settable.</td>
</tr>
<tr>
<td>cd</td>
<td>string</td>
<td>(P*)</td>
<td>Clear to end of display.</td>
</tr>
<tr>
<td>ce</td>
<td>string</td>
<td>(P)</td>
<td>Clear to end of line.</td>
</tr>
<tr>
<td>ch</td>
<td>string</td>
<td>(P)</td>
<td>Horizontal cursor motion only, line stays same.</td>
</tr>
<tr>
<td>cl</td>
<td>string</td>
<td>(P*)</td>
<td>Clear screen.</td>
</tr>
<tr>
<td>cm</td>
<td>string</td>
<td>(P)</td>
<td>Cursor motion.</td>
</tr>
<tr>
<td>co</td>
<td>numeric</td>
<td></td>
<td>Number of columns in line.</td>
</tr>
<tr>
<td>cr</td>
<td>string</td>
<td>(P*)</td>
<td>Carriage return (default (^M)).</td>
</tr>
<tr>
<td>cs</td>
<td>string</td>
<td>(P)</td>
<td>Change scrolling region (VT100), like cm.</td>
</tr>
<tr>
<td>cv</td>
<td>string</td>
<td>(P)</td>
<td>Vertical cursor motion only.</td>
</tr>
<tr>
<td>da</td>
<td>boolean</td>
<td></td>
<td>Display may be retained above.</td>
</tr>
<tr>
<td>dB</td>
<td>numeric</td>
<td></td>
<td>Number of milliseconds of backspace delay needed.</td>
</tr>
<tr>
<td>db</td>
<td>boolean</td>
<td></td>
<td>Display may be retained below.</td>
</tr>
<tr>
<td>dC</td>
<td>numeric</td>
<td></td>
<td>Number of milliseconds of carriage return delay needed.</td>
</tr>
<tr>
<td>dc</td>
<td>string</td>
<td>(P*)</td>
<td>Delete character.</td>
</tr>
<tr>
<td>dF</td>
<td>numeric</td>
<td></td>
<td>Number of milliseconds of formfeed delay needed.</td>
</tr>
<tr>
<td>dl</td>
<td>string</td>
<td>(P*)</td>
<td>Delete line.</td>
</tr>
<tr>
<td>dm</td>
<td>string</td>
<td></td>
<td>Delete mode (enter).</td>
</tr>
<tr>
<td>dN</td>
<td>numeric</td>
<td></td>
<td>Number of milliseconds of newline delay needed.</td>
</tr>
<tr>
<td>do</td>
<td>string</td>
<td></td>
<td>Down one line.</td>
</tr>
<tr>
<td>dT</td>
<td>numeric</td>
<td></td>
<td>Number of milliseconds of tab delay needed.</td>
</tr>
<tr>
<td>ed</td>
<td>string</td>
<td></td>
<td>End of delete mode.</td>
</tr>
<tr>
<td>ei</td>
<td>string</td>
<td></td>
<td>End insert mode.</td>
</tr>
</tbody>
</table>

NOTE: If ic is used, enter: ec=:.
### Table 7-7. termcap Capabilities (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Padding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eo</td>
<td>string</td>
<td></td>
<td>Can erase overstrikes with a blank.</td>
</tr>
<tr>
<td>ff</td>
<td>string</td>
<td>(P*)</td>
<td>Hardcopy terminal page eject (default ^L).</td>
</tr>
<tr>
<td>hc</td>
<td>boolean</td>
<td></td>
<td>Hardcopy terminal.</td>
</tr>
<tr>
<td>hd</td>
<td>string</td>
<td></td>
<td>Half-line down (1/2 linefeed).</td>
</tr>
<tr>
<td>ho</td>
<td>string</td>
<td></td>
<td>Home cursor (if no cm).</td>
</tr>
<tr>
<td>hu</td>
<td>string</td>
<td></td>
<td>Half-line up.</td>
</tr>
<tr>
<td>hz</td>
<td>string</td>
<td></td>
<td>Hazeltime: cannot print tildas (~).</td>
</tr>
<tr>
<td>ic</td>
<td>string</td>
<td>(P)</td>
<td>Insert character.</td>
</tr>
<tr>
<td>if</td>
<td>string</td>
<td></td>
<td>Name of file containing initialization string.</td>
</tr>
<tr>
<td>im</td>
<td>boolean</td>
<td></td>
<td>Insert mode (enter). NOTE: If ic is specified use :im=:</td>
</tr>
<tr>
<td>in</td>
<td>boolean</td>
<td></td>
<td>Insert mode distinguishes nulls on display.</td>
</tr>
<tr>
<td>ip</td>
<td>string</td>
<td>(P*)</td>
<td>Insert pad after character inserted.</td>
</tr>
<tr>
<td>is</td>
<td>string</td>
<td></td>
<td>Terminal initialization string.</td>
</tr>
<tr>
<td>k0-k9</td>
<td>string</td>
<td></td>
<td>Sent by other function keys 0-9.</td>
</tr>
<tr>
<td>kb</td>
<td>string</td>
<td></td>
<td>Sent by backspace key.</td>
</tr>
<tr>
<td>kd</td>
<td>string</td>
<td></td>
<td>Sent by down arrow key.</td>
</tr>
<tr>
<td>ke</td>
<td>string</td>
<td></td>
<td>Take terminal out of keypad transmit mode.</td>
</tr>
<tr>
<td>kh</td>
<td>string</td>
<td></td>
<td>Sent by home key.</td>
</tr>
<tr>
<td>kl</td>
<td>string</td>
<td></td>
<td>Sent by left arrow key.</td>
</tr>
<tr>
<td>kn</td>
<td>numeric</td>
<td></td>
<td>Number of other keys.</td>
</tr>
<tr>
<td>ko</td>
<td>string</td>
<td></td>
<td>termcap entries for other non-function keys.</td>
</tr>
<tr>
<td>kr</td>
<td>string</td>
<td></td>
<td>Sent by right arrow key.</td>
</tr>
<tr>
<td>ks</td>
<td>string</td>
<td></td>
<td>Put terminal in keypad transmit mode.</td>
</tr>
<tr>
<td>ku</td>
<td>string</td>
<td></td>
<td>Sent by up arrow key.</td>
</tr>
<tr>
<td>l0-l9</td>
<td>string</td>
<td></td>
<td>Labels on other function keys.</td>
</tr>
<tr>
<td>li</td>
<td>numeric</td>
<td></td>
<td>Number of lines on screen or page.</td>
</tr>
<tr>
<td>ll</td>
<td>string</td>
<td></td>
<td>Last line, first column (if no cm entry).</td>
</tr>
<tr>
<td>ma</td>
<td>string</td>
<td></td>
<td>Arrow key map.</td>
</tr>
<tr>
<td>mi</td>
<td>boolean</td>
<td></td>
<td>OK to move while in insert mode.</td>
</tr>
</tbody>
</table>
Table 7-7. termcap Capabilities (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Padding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml</td>
<td>string</td>
<td></td>
<td>Memory lock on above cursor.</td>
</tr>
<tr>
<td>ms</td>
<td>boolean</td>
<td></td>
<td>OK to move while in standout and underline mode.</td>
</tr>
<tr>
<td>mu</td>
<td>string</td>
<td></td>
<td>Turn off memory lock.</td>
</tr>
<tr>
<td>nc</td>
<td>boolean</td>
<td></td>
<td>Carriage return down not work.</td>
</tr>
<tr>
<td>nd</td>
<td>string</td>
<td></td>
<td>Non-destructive space.</td>
</tr>
<tr>
<td>nl</td>
<td>string</td>
<td>(P*)</td>
<td>Newline character.</td>
</tr>
<tr>
<td>ns</td>
<td>boolean</td>
<td></td>
<td>Terminal is a non-scrolling CRT.</td>
</tr>
<tr>
<td>os</td>
<td>boolean</td>
<td></td>
<td>Terminal overstrikes.</td>
</tr>
<tr>
<td>pc</td>
<td>string</td>
<td></td>
<td>Pad character (rather than null).</td>
</tr>
<tr>
<td>pt</td>
<td>boolean</td>
<td></td>
<td>Has hardware tabs.</td>
</tr>
<tr>
<td>se</td>
<td>string</td>
<td></td>
<td>End stand out mode.</td>
</tr>
<tr>
<td>sf</td>
<td>string</td>
<td>(P)</td>
<td>Scroll forwards.</td>
</tr>
<tr>
<td>sg</td>
<td>numeric</td>
<td></td>
<td>Number of blank characters left by se or so.</td>
</tr>
<tr>
<td>so</td>
<td>string</td>
<td>(P)</td>
<td>Begin stand out mode.</td>
</tr>
<tr>
<td>sr</td>
<td>string</td>
<td>(P)</td>
<td>Scroll reverse.</td>
</tr>
<tr>
<td>ta</td>
<td>string</td>
<td>(P)</td>
<td>Tab (other than ^I or without padding).</td>
</tr>
<tr>
<td>tc</td>
<td>string</td>
<td></td>
<td>Entry of terminal similar to last termcap entry.</td>
</tr>
<tr>
<td>te</td>
<td>string</td>
<td></td>
<td>String to end programs using cm.</td>
</tr>
<tr>
<td>ti</td>
<td>string</td>
<td></td>
<td>String to begin programs using cm.</td>
</tr>
<tr>
<td>uc</td>
<td>string</td>
<td></td>
<td>Underscore one character and move past it.</td>
</tr>
<tr>
<td>ue</td>
<td>string</td>
<td></td>
<td>End underscore mode.</td>
</tr>
<tr>
<td>ug</td>
<td>numeric</td>
<td></td>
<td>Number of blank characters left by us or ue.</td>
</tr>
<tr>
<td>ul</td>
<td>boolean</td>
<td></td>
<td>Terminal underlines but does not overstrike.</td>
</tr>
<tr>
<td>up</td>
<td>string</td>
<td></td>
<td>Upline (cursor up).</td>
</tr>
<tr>
<td>us</td>
<td>string</td>
<td></td>
<td>Start underscore mode.</td>
</tr>
<tr>
<td>vb</td>
<td>string</td>
<td></td>
<td>Visible bell.</td>
</tr>
<tr>
<td>ve</td>
<td>string</td>
<td></td>
<td>Sequence to end open/visual mode.</td>
</tr>
<tr>
<td>vs</td>
<td>string</td>
<td></td>
<td>Sequence to start open/visual mode.</td>
</tr>
<tr>
<td>xb</td>
<td>boolean</td>
<td></td>
<td>Beehive terminal (f1=&lt;esc&gt;, f2=^C).</td>
</tr>
</tbody>
</table>
Of the capabilities, the most complex and important capability is `cm`: cursor addressing. The string specifying the cursor addressing is formatted similar to the C function: `printf()`. It uses `%` notation to identify addressing encodings of the current line or column position. The line and the column to be addressed could be considered the arguments to the `cm` string. All other characters are passed through unchanged. The following is the notation used for `cm` strings:

Table 7-8. `cm` String Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%d</code></td>
<td>A decimal number (origin 0)</td>
</tr>
<tr>
<td><code>%2</code></td>
<td>Same as <code>%2d</code></td>
</tr>
<tr>
<td><code>%3</code></td>
<td>Same as <code>%3d</code></td>
</tr>
<tr>
<td><code>%</code></td>
<td>ASCII equivalent of value</td>
</tr>
<tr>
<td><code>%+x</code></td>
<td>Adds <code>x</code> to value, then <code>%</code></td>
</tr>
<tr>
<td><code>&gt;%xy</code></td>
<td>If value &gt; <code>x</code> adds <code>y</code>, no output</td>
</tr>
<tr>
<td><code>%r</code></td>
<td>Reverses the order of row and column, no output</td>
</tr>
<tr>
<td><code>%i</code></td>
<td>Increments line/column (for 1 origin)</td>
</tr>
<tr>
<td><code>%%</code></td>
<td>Gives a single <code>%</code></td>
</tr>
<tr>
<td><code>&gt;%n</code></td>
<td>Exclusive or row and column with 0140</td>
</tr>
<tr>
<td><code>%B</code></td>
<td>BCD <code>(16*(x/10) + (x%10)), no output</code></td>
</tr>
<tr>
<td><code>%D</code></td>
<td>Reverse coding <code>(x-2*(x%16)), no output</code></td>
</tr>
</tbody>
</table>
Example String Notations (Continued)

The following examples illustrate the use of the preceding notations:

\texttt{cm=6\textbackslash E\&\%r\%2c\%2Y}

This terminal needs a 6 millisecond delay, rows and columns reversed, and rows and columns to be printed as two digits

\texttt{cm=5\textbackslash E\[%i%d;%dH}

This terminal needs a 5 millisecond delay, rows and columns separated by a semicolon (;), and because of its origin of 1, rows and columns are incremented. The \texttt{\textbackslash esc}[\textbackslash , ; and \texttt{\textbackslash H} are transmitted unchanged. (VT100)

\texttt{cm=\textbackslash E=%+ %+}

This terminal uses rows and columns offset by a blank character. (ABM85H)

Example termcap Entries

\texttt{ka|abm85|kimtron abm85:\}
\texttt{:ce=\ET:cm=\E=%+ %+ :cl=\textasciitilde Z:\}
\texttt{:se=\Ek:so\ Ej:up=\textasciitilde K:sg\#1}

If two entries in the same termcap file are very similar, one can be defined as identical to the other with certain exceptions. To do this, \texttt{tc} is used with the name of the similar terminal. This capability must be the last in the entry. All exceptions to the other terminal must appear before the \texttt{tc} listing. If a capability must be cancelled, use \texttt{<cap>@}. For example, this might be a complete entry:

\texttt{kh|abm85h|kimtron abm85h:\}
\texttt{:se=\EG0:so\EG4:tc=abm85:}
This appendix contains an ASCII conversion chart.
ASCII Symbol Definitions

ASCII is an acronym for American Standard Code for Information Interchange. It consists of 96 printable and 32 nonprintable characters.

The unprintable characters are defined in Table 7-9.

Table 7-9. ASCII Symbol Definitions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>acknowledge</td>
<td>FS</td>
<td>file separator</td>
</tr>
<tr>
<td>BEL</td>
<td>bell</td>
<td>GS</td>
<td>group separator</td>
</tr>
<tr>
<td>BS</td>
<td>backspace</td>
<td>HT</td>
<td>horizontal tabulation</td>
</tr>
<tr>
<td>CAN</td>
<td>cancel</td>
<td>LF</td>
<td>line feed</td>
</tr>
<tr>
<td>CR</td>
<td>carriage return</td>
<td>NAK</td>
<td>negative acknowledgment</td>
</tr>
<tr>
<td>DC</td>
<td>device control</td>
<td>NUL</td>
<td>null</td>
</tr>
<tr>
<td>DEL</td>
<td>delete</td>
<td>RS</td>
<td>record shipment</td>
</tr>
<tr>
<td>DLE</td>
<td>data link escape</td>
<td>SI</td>
<td>shift in</td>
</tr>
<tr>
<td>EM</td>
<td>end of medium</td>
<td>SO</td>
<td>shift out</td>
</tr>
<tr>
<td>ENQ</td>
<td>enquiry</td>
<td>SOH</td>
<td>start of heading</td>
</tr>
<tr>
<td>EOT</td>
<td>end of transmission</td>
<td>SP</td>
<td>space</td>
</tr>
<tr>
<td>ESC</td>
<td>escape</td>
<td>STX</td>
<td>start of text</td>
</tr>
<tr>
<td>ETB</td>
<td>end of transmission</td>
<td>SUB</td>
<td>substitute</td>
</tr>
<tr>
<td>ETX</td>
<td>end of text</td>
<td>SYN</td>
<td>synchronous idle</td>
</tr>
<tr>
<td>FF</td>
<td>form feed</td>
<td>US</td>
<td>unit separator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VT</td>
<td>vertical tabulation</td>
</tr>
</tbody>
</table>
Table 7-10 includes binary, decimal, octal, hexadecimal, and ASCII conversions.

Table 7-10.  ASCII Conversions

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Octal</th>
<th>Hexadecimal</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NUL</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SOH</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>STX</td>
</tr>
<tr>
<td>00000011</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>ETX</td>
</tr>
<tr>
<td>00000100</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>EOT</td>
</tr>
<tr>
<td>00000101</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>ENQ</td>
</tr>
<tr>
<td>00000110</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>ACK</td>
</tr>
<tr>
<td>00000111</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>BEL</td>
</tr>
<tr>
<td>00001000</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>BS</td>
</tr>
<tr>
<td>00001001</td>
<td>9</td>
<td>11</td>
<td>9</td>
<td>HT</td>
</tr>
<tr>
<td>00001010</td>
<td>10</td>
<td>12</td>
<td>A</td>
<td>LF</td>
</tr>
<tr>
<td>00001011</td>
<td>11</td>
<td>13</td>
<td>B</td>
<td>VT</td>
</tr>
<tr>
<td>00001100</td>
<td>12</td>
<td>14</td>
<td>C</td>
<td>FF</td>
</tr>
<tr>
<td>00001101</td>
<td>13</td>
<td>15</td>
<td>D</td>
<td>CR</td>
</tr>
<tr>
<td>00001110</td>
<td>14</td>
<td>16</td>
<td>E</td>
<td>SO</td>
</tr>
<tr>
<td>00001111</td>
<td>15</td>
<td>17</td>
<td>F</td>
<td>SI</td>
</tr>
<tr>
<td>00010000</td>
<td>16</td>
<td>20</td>
<td>10</td>
<td>DLE</td>
</tr>
<tr>
<td>00010001</td>
<td>17</td>
<td>21</td>
<td>11</td>
<td>DC1</td>
</tr>
<tr>
<td>00010010</td>
<td>18</td>
<td>22</td>
<td>12</td>
<td>DC2</td>
</tr>
<tr>
<td>00010011</td>
<td>19</td>
<td>23</td>
<td>13</td>
<td>DC3</td>
</tr>
<tr>
<td>00010100</td>
<td>20</td>
<td>24</td>
<td>14</td>
<td>DC4</td>
</tr>
<tr>
<td>00010101</td>
<td>21</td>
<td>25</td>
<td>15</td>
<td>NAK</td>
</tr>
<tr>
<td>00010110</td>
<td>22</td>
<td>26</td>
<td>16</td>
<td>SYN</td>
</tr>
<tr>
<td>00010111</td>
<td>23</td>
<td>27</td>
<td>17</td>
<td>ETB</td>
</tr>
<tr>
<td>00011000</td>
<td>24</td>
<td>30</td>
<td>18</td>
<td>CAN</td>
</tr>
<tr>
<td>00011001</td>
<td>25</td>
<td>31</td>
<td>19</td>
<td>EM</td>
</tr>
<tr>
<td>00011010</td>
<td>26</td>
<td>32</td>
<td>1A</td>
<td>SUB</td>
</tr>
<tr>
<td>00011011</td>
<td>27</td>
<td>33</td>
<td>1B</td>
<td>ESC</td>
</tr>
<tr>
<td>00011100</td>
<td>28</td>
<td>34</td>
<td>1C</td>
<td>FS</td>
</tr>
<tr>
<td>00011101</td>
<td>29</td>
<td>35</td>
<td>1D</td>
<td>GS</td>
</tr>
<tr>
<td>00011110</td>
<td>30</td>
<td>36</td>
<td>1E</td>
<td>RS</td>
</tr>
<tr>
<td>00011111</td>
<td>31</td>
<td>37</td>
<td>1F</td>
<td>US</td>
</tr>
<tr>
<td>01000000</td>
<td>32</td>
<td>40</td>
<td>20</td>
<td>SP</td>
</tr>
</tbody>
</table>
Table 7-10. ASCII Conversions (Continued)

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Octal</th>
<th>Hexadecimal</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000001</td>
<td>33</td>
<td>41</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>0100010</td>
<td>34</td>
<td>42</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>0100011</td>
<td>35</td>
<td>43</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>0100100</td>
<td>36</td>
<td>44</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>0100101</td>
<td>37</td>
<td>45</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>0100110</td>
<td>38</td>
<td>46</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>0100111</td>
<td>39</td>
<td>47</td>
<td>27</td>
<td>'</td>
</tr>
<tr>
<td>0101000</td>
<td>40</td>
<td>50</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>0101001</td>
<td>41</td>
<td>51</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>0101010</td>
<td>42</td>
<td>52</td>
<td>2A</td>
<td>*</td>
</tr>
<tr>
<td>0101011</td>
<td>43</td>
<td>53</td>
<td>2B</td>
<td>+</td>
</tr>
<tr>
<td>0101100</td>
<td>44</td>
<td>54</td>
<td>2C</td>
<td>,</td>
</tr>
<tr>
<td>0101101</td>
<td>45</td>
<td>55</td>
<td>2D</td>
<td>-</td>
</tr>
<tr>
<td>0101110</td>
<td>46</td>
<td>56</td>
<td>2E</td>
<td>.</td>
</tr>
<tr>
<td>0101111</td>
<td>47</td>
<td>57</td>
<td>2F</td>
<td>/</td>
</tr>
<tr>
<td>0110000</td>
<td>48</td>
<td>60</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>0110001</td>
<td>49</td>
<td>61</td>
<td>31</td>
<td>1</td>
</tr>
<tr>
<td>0110010</td>
<td>50</td>
<td>62</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>0110011</td>
<td>51</td>
<td>63</td>
<td>33</td>
<td>3</td>
</tr>
<tr>
<td>0110100</td>
<td>52</td>
<td>64</td>
<td>34</td>
<td>4</td>
</tr>
<tr>
<td>0110101</td>
<td>53</td>
<td>65</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>0110110</td>
<td>54</td>
<td>66</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>0110111</td>
<td>55</td>
<td>67</td>
<td>37</td>
<td>7</td>
</tr>
<tr>
<td>0111000</td>
<td>56</td>
<td>70</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>0111001</td>
<td>57</td>
<td>71</td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>0111010</td>
<td>58</td>
<td>72</td>
<td>3A</td>
<td>:</td>
</tr>
<tr>
<td>0111011</td>
<td>59</td>
<td>73</td>
<td>3B</td>
<td>;</td>
</tr>
<tr>
<td>0111100</td>
<td>60</td>
<td>74</td>
<td>3C</td>
<td>&lt;</td>
</tr>
<tr>
<td>0111101</td>
<td>61</td>
<td>75</td>
<td>3D</td>
<td>=</td>
</tr>
<tr>
<td>0111110</td>
<td>62</td>
<td>76</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>0111111</td>
<td>63</td>
<td>77</td>
<td>3F</td>
<td>?</td>
</tr>
<tr>
<td>1000000</td>
<td>64</td>
<td>100</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>1000001</td>
<td>65</td>
<td>101</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>1000010</td>
<td>66</td>
<td>102</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>1000011</td>
<td>67</td>
<td>103</td>
<td>43</td>
<td>C</td>
</tr>
<tr>
<td>Binary</td>
<td>Decimal</td>
<td>Octal</td>
<td>Hexadecimal</td>
<td>ASCII</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>1000100</td>
<td>68</td>
<td>104</td>
<td>44</td>
<td>D</td>
</tr>
<tr>
<td>1000101</td>
<td>69</td>
<td>105</td>
<td>45</td>
<td>E</td>
</tr>
<tr>
<td>1000110</td>
<td>70</td>
<td>106</td>
<td>46</td>
<td>F</td>
</tr>
<tr>
<td>1000111</td>
<td>71</td>
<td>107</td>
<td>47</td>
<td>G</td>
</tr>
<tr>
<td>1001000</td>
<td>72</td>
<td>110</td>
<td>48</td>
<td>H</td>
</tr>
<tr>
<td>1001001</td>
<td>73</td>
<td>111</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>1001010</td>
<td>74</td>
<td>112</td>
<td>4A</td>
<td>J</td>
</tr>
<tr>
<td>1001011</td>
<td>75</td>
<td>113</td>
<td>4B</td>
<td>K</td>
</tr>
<tr>
<td>1001100</td>
<td>76</td>
<td>114</td>
<td>4C</td>
<td>L</td>
</tr>
<tr>
<td>1001101</td>
<td>77</td>
<td>115</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>1001110</td>
<td>78</td>
<td>116</td>
<td>4E</td>
<td>N</td>
</tr>
<tr>
<td>1001111</td>
<td>79</td>
<td>117</td>
<td>4F</td>
<td>O</td>
</tr>
<tr>
<td>1010000</td>
<td>80</td>
<td>120</td>
<td>50</td>
<td>P</td>
</tr>
<tr>
<td>1010001</td>
<td>81</td>
<td>121</td>
<td>51</td>
<td>Q</td>
</tr>
<tr>
<td>1010010</td>
<td>82</td>
<td>122</td>
<td>52</td>
<td>R</td>
</tr>
<tr>
<td>1010011</td>
<td>83</td>
<td>123</td>
<td>53</td>
<td>S</td>
</tr>
<tr>
<td>1010100</td>
<td>84</td>
<td>124</td>
<td>54</td>
<td>T</td>
</tr>
<tr>
<td>1010101</td>
<td>85</td>
<td>125</td>
<td>55</td>
<td>U</td>
</tr>
<tr>
<td>1010110</td>
<td>86</td>
<td>126</td>
<td>56</td>
<td>V</td>
</tr>
<tr>
<td>1010111</td>
<td>87</td>
<td>127</td>
<td>57</td>
<td>W</td>
</tr>
<tr>
<td>1011000</td>
<td>88</td>
<td>130</td>
<td>58</td>
<td>X</td>
</tr>
<tr>
<td>1011001</td>
<td>89</td>
<td>131</td>
<td>59</td>
<td>Y</td>
</tr>
<tr>
<td>1011010</td>
<td>90</td>
<td>132</td>
<td>5A</td>
<td>Z</td>
</tr>
<tr>
<td>1011011</td>
<td>91</td>
<td>133</td>
<td>5B</td>
<td>[</td>
</tr>
<tr>
<td>1011100</td>
<td>92</td>
<td>134</td>
<td>5C</td>
<td>\</td>
</tr>
<tr>
<td>1011101</td>
<td>93</td>
<td>135</td>
<td>5D</td>
<td>]</td>
</tr>
<tr>
<td>1011110</td>
<td>94</td>
<td>136</td>
<td>5E</td>
<td>^</td>
</tr>
<tr>
<td>1011111</td>
<td>95</td>
<td>137</td>
<td>5F</td>
<td>_</td>
</tr>
<tr>
<td>1100000</td>
<td>96</td>
<td>140</td>
<td>60</td>
<td>‘</td>
</tr>
<tr>
<td>1100001</td>
<td>97</td>
<td>141</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>1100010</td>
<td>98</td>
<td>142</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>1100011</td>
<td>99</td>
<td>143</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>1100100</td>
<td>100</td>
<td>144</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>1100101</td>
<td>101</td>
<td>145</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>1100110</td>
<td>102</td>
<td>146</td>
<td>66</td>
<td>f</td>
</tr>
</tbody>
</table>
### Table 7-10. ASCII Conversions (Continued)

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Octal</th>
<th>Hexadecimal</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100111</td>
<td>103</td>
<td>147</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>1101000</td>
<td>104</td>
<td>150</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>1101001</td>
<td>105</td>
<td>151</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>1101010</td>
<td>106</td>
<td>152</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>1101011</td>
<td>107</td>
<td>153</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>1101100</td>
<td>108</td>
<td>154</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>1101101</td>
<td>109</td>
<td>155</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>1101110</td>
<td>110</td>
<td>156</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>1101111</td>
<td>111</td>
<td>157</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>1110000</td>
<td>112</td>
<td>160</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>1110001</td>
<td>113</td>
<td>161</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>1110010</td>
<td>114</td>
<td>162</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>1110011</td>
<td>115</td>
<td>163</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>1110100</td>
<td>116</td>
<td>164</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>1110101</td>
<td>117</td>
<td>165</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>1110110</td>
<td>118</td>
<td>166</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>1110111</td>
<td>119</td>
<td>167</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>1111000</td>
<td>120</td>
<td>170</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>1111001</td>
<td>121</td>
<td>171</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>1111010</td>
<td>122</td>
<td>172</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>1111011</td>
<td>123</td>
<td>173</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>1111100</td>
<td>124</td>
<td>174</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>1111101</td>
<td>125</td>
<td>175</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>1111110</td>
<td>126</td>
<td>176</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>1111111</td>
<td>127</td>
<td>177</td>
<td>7F</td>
<td>DEL</td>
</tr>
</tbody>
</table>