

© 1997 Alpha Microsystems

REVISIONS INCORPORATED	
REVISION	DATE

00	March 1988
01	April 1991
02	September 1996
03	May 1997

AMOS System Operator's Guide to the System Initialization Command File
To re-order this document, request part number DSO-000025-00.

This document applies to AMOS 2.3A and later.

The information contained in this manual is believed to be accurate and reliable. However, no responsibility for the accuracy, completeness or use of this information is assumed by Alpha Microsystems.

This document may contain references to products covered under U.S. Patent Number 4,530,048.

The following are registered trademarks of Alpha Microsystems, Santa Ana, CA 92799:

AMIGOS	AMOS	Alpha Micro	AlphaACCOUNTING
AlphaBASIC	AlphaCALC	AlphaCOBOL	AlphaDDE
AlphaFORTRAN 77	AlphaLAN	AlphaLEDGER	AlphaMAIL
AlphaMATE	AlphaNET	AlphaPASCAL	AlphaRJE
AlphaWRITE	CASELODE	OmniBASIC	VER-A-TEL
VIDEOTRAX			

The following are trademarks of Alpha Microsystems, Santa Ana, CA 92799:

AlphaBASIC PLUS	AlphaVUE	AM-PC	AMTEC
AlphaDDE	AlphaConnect	DART	<i>inSight/am</i>
<i>inFront/am</i>	ESP	MULTI	

All other copyrights and trademarks are the property of their respective holders.

ALPHA MICROSYSTEMS
2722 S. Fairview St.
P.O. Box 25059
Santa Ana, CA 92799

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

- 1.1 WHAT IS A SYSTEM INITIALIZATION COMMAND FILE? 1-1
- 1.2 GRAPHICS CONVENTIONS 1-3

CHAPTER 2 - WHAT DOES A SYSTEM INITIALIZATION FILE DO?

- 2.1 SAMPLE SYSTEM INITIALIZATION FILE 2-2

CHAPTER 3 - EDITING YOUR SYSTEM INITIALIZATION FILE

CHAPTER 4 - DEFINING JOBS

- 4.1 DISPLAYING THE INITIALIZATION FILE ON BOOTUP (:T) 4-1
- 4.2 DEFINING THE MAXIMUM NUMBER OF JOBS (JOBS) 4-1
- 4.3 ASSIGNING JOB NAMES (JOBALC) 4-2

CHAPTER 5 - DEFINING TERMINALS

- 5.1 TERMINAL NAME 5-2
- 5.2 INTERFACE-DRIVER 5-2
 - 5.2.1 DISABLING SUPER I/O 5-3
 - 5.2.2 PSEUDO 5-3
- 5.3 TERMINAL 5-4
 - 5.3.1 PSEUDO 5-4
 - 5.3.2 NULL 5-4
 - 5.3.3 ALPHA 5-4
 - 5.3.4 In-width 5-5
 - 5.3.5 In-buffer 5-5
 - 5.3.6 Out-buffer 5-5
- 5.4 EDITOR 5-5
- 5.5 MODEM-DRIVER 5-6
- 5.6 BUILDING YOUR OWN TERMINAL DRIVER 5-6
- 5.7 DEFINING ALTERNATE TERMINAL DRIVERS 5-6
- 5.8 ALLOCATING TERMINAL CONTROL BLOCKS 5-6
 - 5.8.1 The ETHZON Statement 5-7

CHAPTER 6 - SETTING UP YOUR SYSTEM

6.1 SHARED MEMORY ALLOCATION (SMEM)	6-1
6.2 INITIALIZING MEMORY BOARDS (PARITY)	6-1
6.3 THE SCSI DISPATCHER	6-2
6.4 DEFINING DEVICES (DEVTBL)	6-4
6.5 DEFINING DISK BITMAPS (BITMAP)	6-5
6.6 INCREASING THE SYSTEM QUEUE SIZE	6-6
6.7 SETTING UP INTER-TASK COMMUNICATION (MSGINI)	6-7
6.8 SETTING UP THE ERSATZ INITIALIZATION FILE	6-7
6.9 ADDING PROGRAMS TO SYSTEM MEMORY	6-8
6.9.1 Including Device Drivers in System Memory	6-10
6.9.1.1 Setting Up Write Buffering	6-10
6.9.2 Defining Floppy Disk Drivers to Use Buffered I/O	6-11
6.9.3 Front Panel Write Buffer Display	6-12
6.9.4 Enabling Interpreted Prompts	6-12

CHAPTER 7 - FINISHING THE SYSTEM INITIALIZATION

7.1 SETTING OPTIONS	7-1
7.2 SETTING UP JOBS	7-2
7.2.1 Attaching Terminals to Jobs	7-2
7.2.2 Initializing Jobs (KILL)	7-3
7.2.3 Forcing Input to Jobs	7-4
7.2.4 Allocating Memory to Jobs	7-4
7.2.5 The VER Command	7-5
7.3 MOUNTING DISKS	7-5
7.4 SETTING UP EVENT LOGGING	7-5
7.5 SETTING UP THE TASK MANAGER	7-6
7.6 SETTING UP THE PRINT SPOOLER	7-6
7.7 SETTING UP ALPHACD	7-7
7.8 FORCING THE OPERATOR JOB TO RUN TASKS	7-7
7.9 MEMORY 0	7-7

CHAPTER 8 - TESTING YOUR SYSTEM INITIALIZATION FILE

8.1 THE MONTST	8-1
8.2 RENAMING YOUR TEST FILE	8-1
8.3 CREATING A MINIMAL FILE	8-2

APPENDIX A - SAMPLE SYSTEM INITIALIZATION COMMAND FILE**APPENDIX B - GLOSSARY**

AMOS COMMAND LEVEL	B-1
AMOS PROMPT	B-1
BASIC	B-1
BINARY	B-1
COMMAND LINE	B-1
COMMAND FILE	B-1
CONTROL SEQUENCE	B-2

DEFAULT	B-2
DELIMITER	B-2
DISK DRIVE CONTROLLER	B-2
DISK FILE	B-2
DSK0:	B-2
ECHO	B-3
FILE LOCKING	B-3
FILE SPECIFICATION	B-3
INITIALIZATION	B-3
INPUT	B-3
JOBS	B-3
LOGICAL DEVICE	B-3
MOUNT	B-4
NON-SELF-CONFIGURING DISK	B-4
PHYSICAL ADDRESS	B-4
PHYSICAL UNIT/DEVICE	B-4
QUEUE	B-4
RE-ENTRANT	B-4
RE-USABLE	B-4
SAVE	B-4
SELF-CONFIGURING DISK	B-4
SEQUENTIAL	B-5
SPOOLING	B-5
SWITCH or OPTION	B-5
SYSTEM MEMORY	B-5
USER MEMORY	B-5
USER NAME	B-5
WILDCARD	B-6
WILDCARD FILE COMMAND SWITCHES	B-6

CHAPTER 1

INTRODUCTION

Almost every function of the System Operator requires you to be familiar with the system initialization command file. This manual introduces you to this most important part of your computer system. It answers these general questions:

- What is the system initialization command file?
- What makes up the system initialization command file?
- How do you modify the file to customize your system?

1.1 WHAT IS A SYSTEM INITIALIZATION COMMAND FILE?

The greatest strength of the Alpha Micro computer system is its flexibility. An Alpha Micro computer system is "device independent"; that means you are not locked into using a particular type of disk drive or terminal. In addition, you are not locked into a pre-set configuration of jobs and memory partitions. This flexibility lets you set up your computer according to whatever hardware you have, and to tailor both hardware and software to be most efficient for your needs.

Alpha Micro computers also provide a simple way to change the components of your computer at a later date when the needs of the users on your computer change.

The heart of this flexible system is the system initialization command file, the mechanism by which you configure the Alpha Micro operating system (called the "monitor") to fit the particular combination of hardware devices you have connected to your computer.

This special file, which is named different things on different types of AMOS systems (for example, AMOSL.INI on AMOS/L systems, or AMOS32.INI on AMOS/32 systems), defines all of the devices on your computer to the monitor and lets it communicate with those devices in the particular manner required by their characteristics.

The system initialization file also gives other important information to the monitor about the setup of the jobs on your computer and the allocation of system resources to those jobs.



A "command file" is a text file that contains system commands. When you call the command file, the monitor reads and obeys the instructions contained in the file.

Each time the system is powered up or reset, the Alpha Micro hardware causes the monitor program—which has the system name and a .MON extension—for example, AMOSL.MON), to be loaded into memory and executed. The monitor reserves a small memory partition at the far end of memory for processing the system initialization command file. Then, the monitor loads the system initialization command file into that area of memory and starts processing it.

Some of the lines in the system initialization command file represent one system function or parameter which determines the characteristics of the monitor. Others set up various aspects of your computer. The monitor checks the system initialization command file to find out what devices are on the computer and what special programs and functions you want to add to the your system memory. As the monitor reads in a line from the system initialization file, it loads in the specified program from DSK0:[1,4] and runs it.

The first thing the system initialization command file does is define one or more jobs; next, it defines one or more terminals. The monitor automatically attaches the first job defined (called the "Operator Job") and the first terminal defined (called the "Operator Terminal"), and then proceeds to process the rest of the system initialization command file under the control of that job in the temporary memory partition previously allocated.

The monitor executes each command line as it would a command in any other command file. However, because this command file is the system initialization command file, the monitor performs some of the commands differently than it would the same commands after the computer is up and running. The execution of certain commands (such as JOBS, TRMDEF, DEVTBL, etc.) performs the actual system generation.

The command that finishes the system initialization is "MEMORY 0"; this tells the monitor to free the temporary memory partition the system initialization command file was processed under and give the Operator Job all available memory not already assigned to other jobs.

Once the monitor has finished processing the system initialization command file, your computer is up and running. We call this entire process, from the time you turn on your computer or push the reset button until Ct is up and running, "bootup."











You can think of the AMOS monitor file as being a skeleton monitor—in the process of executing the system initialization command file, the monitor adds to itself in memory by filling in various tables in memory with information on terminals, disks drives, and other devices attached to the computer. Only by processing the information in the system initialization command file can the monitor find out the exact configuration of your hardware, and what jobs you want to run on your computer.

1.2°°GRAPHICS CONVENTIONS

This manual conforms to the other Alpha Micro publications in its use of a standard set of graphics conventions. We hope these graphics simplify our examples and make them easier for you to use.

Any example that shows you pressing the `RETURN` key (see below) is a command entered at AMOS command level. We do not show an AMOS prompt symbol in these examples. The AMOS prompt defaults to a period (.), but may be set to any symbol or group of up to 19 characters. See the SET reference sheet in your *System Commands Reference Manual* for information on setting the AMOS prompt.

SYMBOL	MEANING
devn:	Device-Name. The "dev" is the three letter physical device code, and the "n" is the logical unit number. Examples of device names are DSK0:, DSK5:, WIN1:, and MTU0:. Usually, device names indicate disk drives, but they can also refer to magnetic tape drives and video cassette recorders.
filespec	File Specification. A file specification identifies a specific file within an account. A complete filespec is made up of the devn:, the filename, the file extension, and the project-programmer number. For example: <div style="text-align: center;"> devn:filename.ext[p,pn] -or- DSK0:SYSTAT.LIT[1,4] </div>
[p,pn]	This abbreviation represents an account on a disk you can store files and data in. An actual disk account number looks like this: [100,2] or [1,4].
TEXT	Bold text in an example of user/computer communication represents the characters you type.
TEXT	Text like this in an example of user/computer communication represents the characters the computer displays on your terminal screen.
<code>KEY</code>	In our examples, the keycap symbol appears whenever you need to press a certain key on your terminal keyboard. The name of the key you need to press appears inside the keycap symbol, like this: <code>RETURN</code> . If you need to press the TAB key, you would see <code>TAB</code> , or the ESCAPE key, <code>ESCAPE</code> . (Sometimes the ESCAPE key is labeled ESC or ALT MODE.)

SYMBOL	MEANING
{ }	<p>Braces are used in some examples to indicate optional elements of a command line. In the example:</p> <pre data-bbox="553 409 764 436">DIR{/switch}</pre> <p>the braces tell you /switch is not a required portion of the DIR command line.</p>
/	<p>The slash symbol precedes a command line switch or "option request." For example:</p> <pre data-bbox="553 682 787 709">DIR/WIDE:3 </pre> <p>This command requests a directory display of the disk account you are currently logged into. The switch (/WIDE:3) indicates you want the display to be three columns wide.</p>
 / 	<p>This indicates a control sequence you press on the keyboard. The  key is pressed and held down while the indicated key is also pressed.</p>
^	<p>This symbol in front of a capital letter means the letter is a "control character." For example, when you press /, it appears on your screen as ^C. (^C is the control character used like the  key to cancel most programs and return you to AMOS command level.)</p>
	<p>This symbol means "halt!" It indicates an important note you should read carefully before going further in the documentation. Usually, text next to this symbol contains instructions for something you MUST or MUST NOT do, so read it carefully.</p>
	<p>This symbol means "hint." It indicates a helpful bit of information, or a "short cut" that could save you time or trouble.</p>
	<p>This symbol means "remember." It indicates something you should keep in mind while you are following a set of instructions.</p>

CHAPTER 2

WHAT DOES A SYSTEM INITIALIZATION FILE DO?

Later chapters will treat each of the points below in detail. Briefly, however, the system initialization command file does the following:

- Defines how many jobs will run on your computer, and gives them names.
- Performs any initialization required by the memory boards in your computer.
- Defines the terminals on your computer to the monitor by identifying what Input/Output interface boards the terminals are connected to, what terminal driver and interface driver programs to use for each terminal, what baud rate the terminals use, etc.
- Defines the disk drives on your computer by telling the monitor what driver programs to use, defining bitmaps for the devices, and arranging physical unit/logical device combinations.
- Defines any special devices on your computer (such as video cassette recorders, serial printers, etc.), telling the monitor what device driver programs to use for the devices, and whether the devices may be shared.
- Tells the monitor what to load into system memory.
- Tells the monitor what terminals to attach to which jobs.
- Allocates memory to jobs.
- Performs optional procedures such as setting up printer spoolers, running special programs, mounting disks, etc.

2.1 SAMPLE SYSTEM INITIALIZATION FILE

```

:T
JOBS 3 ; Define number of jobs
TRMDEF CRT0,AM100L=0,ALPHA,80,80,80,EDITOR=5
;
PARITY ; Error detection on
TRMDEF PRINTR,AM100L=1,ALPHA,80,80,80 ; Define printer
TRMDEF MANAGR,PSEUDO,NULL,25,25,25 ; Define Task Manager
TRMDEF SRVTTM,PSEUDO,NULL,100,100,100 ; Task Manager job
TDVDEF * ; Define alternate terminal
; drivers
JOBALC OPR,TASK,SERVNT ; Define job names
;
DEVTBL DSK1 ; Define disk
DEVTBL /VCR0 ; Define VCR interface
BITMAP DSK ; Set up disk bitmap
ERSATZ ERSATZ.INI ; Define Ersatz names
MSGINI 16K
;
SYSTEM SYMSG.USA ; Set USA messages
SYSTEM TRM.DVR ; Load drivers in
SYSTEM VCR.DVR ; system memory
SYSTEM QFLOCK.SYS ; Needed for TM
SYSTEM
;
LOG SYS:
SYSTEM SERVICE ; User name for logon
;
SET DSKERR ; Report disk errors
MOUNT DSK1: ; Mount the disk.
;
SETJOB MANAGR,SERVNT,64K ; Set up TM servant job
ATTACH SRVTTM,SERVNT
;
ATTACH MANAGR,TASK ; Set up Task Manager
KILL TASK
FORCE TASK
MEMORY 28K
LOG SYS:
TSKINI
B SERVNT.INI ; Initialize SERVNT
S PRINTR.INI ; Initialize printer
G

; Blank line above ends FORCE
WAIT TASK
MEMORY 0 ; Set operator's memory

```

CHAPTER 3

EDITING YOUR SYSTEM INITIALIZATION FILE

Your computer was delivered to you with a system initialization command file configured for the particular hardware that made up your initial computer.

However, you will probably want to set your computer up in your own way by adding jobs, changing memory allocations, and so on.

In addition, if you want to add a new piece of hardware, you will need to modify your system initialization command file to define the new device to the monitor. Or, you may decide you want to load certain programs (such as BASIC) into system memory so all users can use that copy instead of everyone loading it into his or her own memory partition. Or, you may want to tell the monitor to run particular programs whenever the computer boots.



It is very important you take special care when modifying the system initialization command file. If you change a system initialization command file that works, and somehow your new system initialization command file doesn't work, you won't be able to get the computer up and running off that disk. Therefore, never modify the system initialization command file directly—copy it and modify the copy!

Since the size of the system job that runs the system initialization command file is fixed, it is a good idea to segment your system initialization file into three or more files. That is, have your main system initialization file call other .CMD or .DO files to process parts of the system initialization set-up. This makes it easier to organize the set-up and makes sure a large .INI file will not overload the system. Breaking the files into only two will not help, as the calling file and the called file temporarily share the memory space, and hence might be too large. As an example, you might have an AMOSL.INI file that ends with:

```
JOBSET
```

Which calls the JOBSET.CMD file. JOBSET.CMD might then call TRMSET.CMD, etc.

Alpha Micro provides a program called MONTST that allows you to test your new system initialization command file to make sure it works properly, without the danger of being unable to boot. The only way to boot from a system initialization command file that is NOT named with the default name of your system initialization command file is by using MONTST.

If you did modify the file directly, and made a mistake, before you could again use the disabled disk as a System Disk, you would need to bring the computer up off of another System Disk or warm boot tape, and transfer over a copy of a good system initialization command file.

If we seem overly cautious in handling the system initialization command file, just remember—**When you press the RESET button, the computer will not boot if your system disk does not contain a good system initialization command file!**

Remember, too, the system initialization command file must have the proper name and extension (for example, AMOSL.INI or AMOS32.INI).

This section discusses the procedures you should follow when modifying a system initialization command file.

Before you begin, it is a good idea to make a bootable backup copy of your System Disk.

The first step is to make a copy of your system initialization command file. In this example, we'll call it TEST.INI:

```
COPY TEST.INI=AMOS32.INI 
```

Now, edit the TEST.INI file using the AlphaVUE text editor:

```
VUE TEST.INI 
```



When making a number of changes to the TEST.INI file, it is wise to test your changes in steps. For example, if you are going to add two new terminals, add a disk device, and change job memory allocations, do just one set of changes at a time. For example, add the new terminals and then test the TEST.INI file. If everything works well, then add the disk device. After successfully testing the TEST.INI file again, make your memory allocation changes. In other words, keep the number of changes to a minimum. Then if you make an error, you won't have to wonder which of the many changes you just made caused the problem.

The next chapters will take you through the editing process, explaining how to change each area of the system initialization command file.

CHAPTER 4

DEFINING JOBS

This chapter tells you how you set up jobs on your computer. A "job" is a "workplace" in memory that has a name for identification purposes, and has a terminal attached to it.

4.1°°DISPLAYING THE INITIALIZATION FILE ON BOOTUP (:T)

The first line in a system initialization command file is usually :T. This tells the monitor to display the system initialization command file on a terminal as it is processing.

When the monitor first finds the system initialization command file, the process of reading and processing the file is initially under control of the first job defined (to which no terminal has yet been attached). As soon as a terminal is defined (by the first TRMDEF command), the monitor displays the rest of the system initialization command file on that terminal screen as it executes the file.

If you don't want the monitor to display the system initialization command file as it processes it, replace the :T with :S. If you only wish to see error messages if they appear, or FORCED program input, replace the :T with a :R.



The semicolons in the examples indicate comment lines. The monitor does not process comment lines, but does display them as it processes the system initialization command file.

4.2°°DEFINING THE MAXIMUM NUMBER OF JOBS (JOBS)

The JOBS command tells the monitor how many jobs to allocate on the computer. This defines how many entries are made in the job table within the system monitor.

You must always specify at least one job to complete the system initialization process. This first job is used to execute the remainder of the system initialization command file. We call this the "Operator Job."

You may allocate as many jobs as you wish with the JOBS command, though it can be wasteful of memory if you define many more jobs than you need. Each job you specify will cause four bytes of memory to be reserved within the monitor area of memory. No additional space is used until a job is named by using the JOBALC command.



Specify an extra job for the use of the BACKUP command in your JOBS statement.

At AMOS command level, the JOBS command tells you how many jobs have been allocated, and how many have been attached to terminals.

4.3^{oo}ASSIGNING JOB NAMES (JOBALC)

Once the maximum number of jobs is defined (by JOBS), you may assign names and allocate JCB's (Job Control Blocks) to those jobs you wish to use.

The first job listed in the JOBALC command line is the Operator Job, which processes the system initialization command file when the computer is reset.

You may give the jobs any name you choose up to six characters long, except the first character must not be a number, as this may confuse certain programs.

You may have as many JOBALC commands as you wish, as long as they are before the final SYSTEM command.

Each job assigned by a JOBALC command adds approximately 1750 bytes to the operating system. You may not use JOBALC to assign more jobs than have been defined in the JOBS command.

The first job listed in the JOBALC command line and the terminal defined in the first TRMDEF statement are automatically attached; the computer comes up under that job and you see the system initialization command information displayed on that terminal. However, except for the first job and terminal, the JOBALC command does not associate a terminal with a job.

CHAPTER 5

DEFINING TERMINALS

The TRMDEF command defines a terminal to AMOS. Every terminal has a name (1 to 6 characters), a specific hardware interface, and a terminal driver (a program that does any necessary character conversions). The TRMDEF command also specifies the size of the various buffers used in the data transfers between the terminal and the computer. The TRMDEF command takes this form:

```
TRMDEF Name,Interface-driver,Terminal,In-width,In-buffer,  
Out-buffer{{,EDITOR}{=Editor-buffer}}{,Modem-driver}
```

When the monitor processes a TRMDEF command line, it builds a terminal definition unit in system memory which includes all of the elements above. The computer loads in the correct terminal driver and interface driver and links them to the definition unit; then it executes the interface driver which performs any necessary interface initialization. When this process is complete, your terminal is ready for use with the computer.



The buffer size values in TRMDEF command lines affect the total size of the monitor—be careful not to use too much memory space for buffers. How much is too much depends on your computer and memory.

After the monitor has finished processing the system initialization command file, the TRMDEF command performs a different function. After the computer is up and running, TRMDEF becomes a user command that displays the current terminal configuration of the computer in a form similar to the original TRMDEF command lines in the system initialization command file. The octal number following each terminal name is the absolute address in the monitor of the terminal definition unit for that terminal (this information is sometimes useful when debugging the terminal service system; in general, you can ignore it). See your *System Commands Reference Manual* for information.



If you physically connect a terminal to your computer without defining it with a TRMDEF statement in your initialization file, problems can result when MONTSTing. Any input from the terminal interferes with the computer, since AMOS was not told there should be signals coming in from that port. This interference could cause annoying problems or even system "crashes."

A sample TRMDEF command line might look like this:

```
TRMDEF TERM1,AM355=1,ALPHA,100,100,100,EDITOR=3
```

Now we'll discuss the different elements of the TRMDEF command line:

5.1 °TERMINAL NAME

The terminal name consists of one to six alphanumeric characters chosen by you. Every terminal on the computer must have a different name, although you may choose to use a terminal name that duplicates a job name or a program name. The computer uses the terminal name to identify the terminal you want to attach to a job or you want to access using the TRM device driver.

5.2 °INTERFACE-DRIVER

The interface is the hardware board connecting the terminal to the computer. The interface statement gives the name of the terminal interface and the terminal's I/O port number on the computer. The I/O port address follows the name of the terminal interface, and is separated from it by an equal sign.

As the computer processes each TRMDEF command line, it loads the proper interface driver into system memory from account [1,6] of the System Disk. If the driver is already in memory because of a previous TRMDEF command line, the computer does not load it in again.

Interface drivers are the programs that actually transfer data between the terminal buffers and the terminal interface boards; these programs have the extension .IDV and must be in account [1,6] of the System Disk. The interface drivers often have the same name as the interface boards they work with. The format of the interface section of the TRMDEF command line is:

```
Interface-driver-name = I/O-port-number{:baud rate}
```

For example:

```
AM355 = 1:9600
```

The default baud rate is 19200. That is, if you do not specify a baud rate, the computer will use 19200 as the baud rate.

The I/O port number tells the interface driver where the terminal is connected to the specific board defined by Interface-driver-name. For example, the AM100L driver controls the two ports defined on the AM-100/L CPU board. Therefore, you can connect two terminals—one to port 0, and one to port 1.

The port number you specify in the TRMDEF statement may not always be the same one as the number of the "port" in the back panel of your chassis where the terminal cable plugs in. Therefore, we call those back-panel ports "external" ports.

Each interface board uses one or more specific interface drivers. Please refer to your computer *Owner's Manual* or the instructions on installing your interface board for the interface driver name.

5.2.1 [∞]DISABLING SUPER I/O

In AMOS 2.3 and later, Super I/O is enabled by default for all I/O boards which support it, such as the AM-318 and AM-359. However, if you have an application program that does not run properly on a Super I/O-enabled serial port, you can disable Super I/O.

To disable Super I/O for all ports on all boards which use a particular interface driver, add the /O switch (the letter "O," not a zero) after the interface driver name on the first TRMDEF statement using that interface driver. For example:

```
TRMDEF TERM2,AM318/O=0:19200,AM65,100,100,100
```

You must add /O to the **first** TRMDEF using this interface driver; if you put it on any other statement, it will have no effect. It turns off Super I/O on all ports using that driver; you cannot turn off Super I/O for some ports while leaving it on for others.

However, if you have more than one board using the same interface driver (for example, multiple AM-359 boards), with a little extra effort you can turn Super I/O off for some boards while leaving it on for others. Here's the procedure:

1. [∞]Log to the DVR: account and make a copy of the interface driver file under another name. For example:

```
COPY AM359N.IDV=AM359.IDVRETURN
```

2. [∞]In your initialization file, locate the TRMDEF statements for the ports on the boards where you want to disable Super I/O. For all of the TRMDEFs, replace the standard interface driver name with the new driver name (in our example, AM359N). Do this only for ports on the boards for which you want to disable Super I/O.

3. [∞]Add the /O switch to the first TRMDEF statement that uses the new (AM359N) driver, as described above. For example:

```
TRMDEF TERM17,AM359N/O=20:19200,AM65,100,100,100
```

The /O switch will disable Super I/O only for the boards using the copy of the interface driver. All I/O boards using the standard driver will remain Super I/O enabled.

5.2.2 [∞]PSEUDO

One special interface driver deserves mention. You normally use the PSEUDO driver with either the PSEUDO or NULL terminal specifications in a TRMDEF command line. This sets up a software interface driver that communicates with a pseudo terminal for those occasions when you have a job that doesn't need a real, hardware-controlled terminal for processing (such as a printer spooler job). The PSEUDO interface driver is built into the monitor, and does not reside in account [1,6] of the System Disk.

5.3°TERMINAL

The terminal statement tells the computer what kind of terminal is connected to the interface board, and thus what kind of terminal driver to load into system memory from account [1,6] of the System Disk. Different terminals process characters differently.

A terminal driver is the program that does the necessary code conversion and character processing required by the particular terminal that it supports. It is the terminal driver, then, that takes care of the special functions (for example, cursor control, Control-U, rubout, null characters after RETURNS, etc.) that differ between terminal types.

Terminal drivers have the extension .TDV and are sharable; that is, a given driver is loaded only once into system memory, no matter how many terminals of the same type are defined. Some of the terminal drivers currently available on the computer are:

5.3.1°PSEUDO

This is the driver for the software-controlled pseudo terminals. PSEUDO.TDV merely stops echoing of input characters and allows buffering of input to and from the controlled job. Use it only with the PSEUDO interface statement. The PSEUDO terminal driver is built into the monitor, and is not in the [1,6] account of the System Disk.

5.3.2°NULL

This driver is identical to the PSEUDO driver, except it discards the terminal output from the job, instead of buffering it to wait for some other job to pick it up. Use this terminal driver when you want to control a job whose terminal output is of no importance (such as the printer spooler).

When using this driver for a job like the printer spooler, you will usually use the FORCE command to send commands and data to that job; make sure the buffer sizes you define in the pseudo-terminal's TRMDEF statement (see below for information on buffers) are large enough to accept the lines of data you are going to FORCE to the job.

5.3.3°ALPHA

The ALPHA.TDV terminal driver supports the Alpha Micro AM-60 and compatible terminals. An Alpha Micro AM-60 compatible terminal is any terminal using the same internal codes as an AM-60. See your dealer for terminal compatibility information.

5.3.4[∞]In-width

The in-width statement specifies the maximum terminal line-width allowed before a carriage return. Allowing a large width, such as 100, gives an added margin of safety when typing long lines.

5.3.5[∞]In-buffer

There are times when the computer cannot immediately process characters you type from the keyboard. Instead, it stores the characters in an input buffer until it can get around to them. The in-buffer statement specifies the size of this buffer. The number you specify, then, is also the number of characters you can type ahead of the computer before it starts to discard characters. When you've reached the end of the type-ahead buffer, the computer echoes any additional characters as bell codes and discards them. If you want to be able to type ahead a full line, make this parameter at least as large as the in-width value.

5.3.6[∞]Out-buffer

The out-buffer statement specifies the size of the terminal output buffer. This is the buffer holding characters the computer sends to the terminal. The terminal empties this buffer at its own speed. The computer allocates two output buffers of the size specified in the out-buffer statement. It allows a job to stay active until it fills these buffers; then the job is put into the terminal output wait state. In general, specify larger output buffers for faster terminals, and specify smaller output buffers (perhaps only ten characters or so) for slower terminals.

For more details on these input and output buffers, see the section on the terminal service system in the AMOS Monitor Calls Manual



The monitor size is increased as buffers are increased.

5.4[∞]EDITOR

The EDITOR statement activates the AMOS command line editor, which allows you to edit your command lines at AMOS command level. If you include an equal sign followed by a number, the line editor will also allow you to "call back" previous command lines for re-editing and/or re-execution. The number specified tells AMOS how many of the previous command lines you would like to be able to recall—for example, EDITOR=5 would allow you to call back any of the last five commands you entered. When you define the EDITOR, remember each "call back" line uses memory space, so you may not want to define more lines than you normally use.



If I/O Redirection is to be used with the AMOS line editor, you must not have a TRMDEF statement with buffers larger than 200 bytes or specifying more than 20 command lines to recall.

5.5°MODEM-DRIVER

If you want the terminal defined to be able to work with a modem (using telephone lines to transmit data), enter the name of the modem driver program at the end of your TRMDEF line. If you have not specified the EDITOR option, remember to include an extra comma as a placeholder. Some examples:

```
TRMDEF TERM1,AM355=1,ALPHA,100,100,100,EDITOR=3,MODEM.DVR
TRMDEF TERM1,AM355=1,ALPHA,100,100,100,,MDM2.DVR
```

5.6°BUILDING YOUR OWN TERMINAL DRIVER

If you want to add a terminal to your Alpha Micro computer system not compatible with a type of Alpha Micro terminal, you will need to build your own terminal driver for that device. You can use Alpha Micro terminal driver source files as guides. See the *AMOS Terminal Programmer's Guide* for more information.

5.7°DEFINING ALTERNATE TERMINAL DRIVERS

You can define alternate terminal drivers to the computer. These alternate terminal drivers may be selected from AMOS command level by using the SET TERMINAL DRIVER command. This allows you to switch the driver for your terminal without changing your initialization file or rebooting your computer. The TDVDEF program defines alternate terminal drivers to your computer. Enter a TDVDEF command followed by the names of the terminal drivers you would like to have defined. For example:

```
TDVDEF ALPHA,AM60,TV1920
```

The default extension is .TDV, and the default disk/account is DSK0:[1,6]. If the driver has already been defined, the new definition is ignored. You can load all of the .TDV files from DSK0:[1,6] by entering:

```
TDVDEF *
```



All TDVDEF commands MUST come before the first SYSTEM command.

5.8°ALLOCATING TERMINAL CONTROL BLOCKS

You can use a special form of TRMDEF to allocate a pool of Terminal Control Blocks (TCBs) or Terminal Definition Units to be used by jobs "spawned" by VTSER and other similar networking processes. This should be the last TRMDEF statement in your initialization file. The special command syntax is:

```
TRMDEF #tcb-number, In-width, In-buffer, Out-buffer
      {{,EDITOR}{=Editor-buffer}}{,Modem-driver}
```

where **tcb-number** is the number of TCBs to allocate. All the other parts of the line are the same as in normal TRMDEF statements. For example:

```
TRMDEF #10 100,100,100,EDITOR=10
```

would allocate 10 TCBs for use by remote job connections, each with buffers 100 bytes long. Also, the line editor is enabled for all TCBs, with 10 recall buffers available. The number of TCBs allocated should be equal to the greatest number of remote connections on your system.

Before allocating the TCBs, TRMDEF attempts to load ETHNET.IDV from DSK0:[1,6]. If it is not there, you see an error message and TRMDEF aborts.

The number of TCBs you can allocate depends on your AMOS user license. The total number of actual terminals defined (not including pseudo terminals) plus the TCB pool cannot be greater than your licensed user count. If you attempt to allocate more TCBs than you have available, AMOS will allocate the maximum number allowable under your user license and display this message:

```
Requested pool count of xx changed to a count of yy
```

xx is the number of TCBs specified in the TRMDEF statement; **yy** is the actual number created.

You can have only one TRMDEF #tcb-number statement in your initialization file. If you enter more than one, all such statements after the first one will be ignored. You'll see this error message:

```
Multiple TCB pool requests not allowed
```

5.8.1^oThe ETHZON Statement

Immediately after the TRMDEF #tcb-number statement, you should add an ETHZON statement for the same number of spawned network jobs. The format is:

```
ETHZON jobs
```

Jobs is the maximum number of jobs you will need to allocate for Type 2 virtual connections. It should be the same as the number of TCBs allocated with TRMDEF. For example:

```
TRMDEF #20 100,100,100,EDITOR=5  
ETHZON 20
```

CHAPTER 6

SETTING UP YOUR SYSTEM

This chapter discusses the commands you use to get your computer running, to set up memory, and to customize your system memory with the programs you use most often.

6.1 SHARED MEMORY ALLOCATION (SMEM)

The SMEM command allocates a shared memory pool for programs. The format is:

```
SMEM size{K}{M}
```

size is the amount of memory to allocate, **K** is Kilobytes, and **M** is Megabytes.



You may put one and only one SMEM command in your .INI file. It must immediately follow the last SYSTEM command. SMEM uses three queue blocks (and so must come after the QUEUE command).

Here are some examples of SMEM lines:

```
SMEM 20000  
SMEM 70K  
SMEM 10M
```

The SMEM command at AMOS command level displays information about the shared memory pool, and lets you remove "stuck" blocks. See the *System Commands Reference Manual* for more information.

6.2 INITIALIZING MEMORY BOARDS (PARITY)

The memory boards in your computer require the presence of the PARITY command in your system initialization command file to turn on parity error detection and reporting.

The sample system initialization command file in Chapter Two shows the PARITY command being used after the first TRMDEF. Placing the command here activates parity error detection during the system initialization command file process and displays parity error messages on the Operator Terminal.

6.3 THE SCSI DISPATCHER

All newer Alpha Micro computers (AM-4000, Roadrunners, Eagles, and later) use a SCSI dispatcher program to communicate with the SCSI controller chip.

For each type of computer, there are two versions of the SCSI dispatcher: the "simple" dispatcher and the "enhanced" (PIC-encoded) dispatcher. The simple dispatcher is only intended to initially bring up a new or upgraded system; to make a warm-boot tape; or for temporary situations on computers which do not have an SSD chip. It does not support enhanced SCSI functions and is **not meant for normal system operation**. The enhanced SCSI dispatcher supports additional SCSI features such as command queueing, synchronous transfers, and multi-threaded and scatter-gather operations. The enhanced dispatcher provides a tremendous performance increase over the simple dispatcher. The SCSI dispatcher supports up to seven SCSI devices on the same bus, or fifteen devices on a Wide-SCSI bus.



The enhanced dispatcher uses the same PIC code as the *AMOS monitor*. Once you enter the monitor PIC code to enable the requested number of users, and define the enhanced dispatcher in your system initialization file, the dispatcher is also enabled.

If you have *any* SCSI peripherals attached to the SCSI port on a dispatched computer, you **MUST** define the SCSI dispatcher in your boot INI, regardless of whether you are using SCSI-1 or SCSI-2 peripheral devices.

The name of the dispatcher depends on the type of computer:

System Type	Enhanced Dispatcher	Simple Dispatcher
Roadrunner 030 and 040, Eagle 100-550	SCZRR	SIMRR
AM-4000, AM-4000M, and AM-3000 with AM-540	SCZ190	SIM190
Roadrunner 060, AM-6000	SCZR60	SIMR60

To define the SCSI dispatcher in your system initialization file enter:

```
SCZDSP xxxxxx.SYS
```

xxxxxx is the name of the dispatcher version for your computer, as stated above. For example:

```
SCZDSP SCZRR.SYS ;Enable enhanced SCSI dispatcher
```

The SCZDSP statement must come after the last TRMDEF statement and before the first DEVTBL statement.

The SCZRR and SCZ190 dispatchers accept one command line switch, which is used mainly in troubleshooting situations. As stated previously, the enhanced dispatchers support synchronous data transfer. Under normal conditions the dispatcher attempts to communicate with the drive in synchronous mode, and, if unsuccessful, switches to non-synchronous mode automatically. However, you can force these dispatchers to

operate in non-synchronous mode only by entering a /N at the end of the command line. For example:

```
SCZDSP SCZ190.SYS/N
```

The SCZR60 dispatcher accepts several switches:

Switch	Meaning
/EW{id#}	Enable wide SCSI operation for all devices or for just the device at SCSI ID <i>id#</i> .
/NQ{id#}	Disable command queuing for all devices or for just the device at SCSI ID <i>id#</i> .
/NS{id#}	Disable synchronous negotiation for all devices or for just the device at SCSI ID <i>id#</i> . This is the same as the /N switch for the SCZRR and SCZ190 dispatchers, with the addition of the device number option.
/NP	Disable parity checking for all devices. Parity is still generated. This option always affects all devices on the bus; you cannot specify a device ID.

The most common of these switches is /EW, to enable wide SCSI operation when using the optional wide SCSI bus. For example, if you have a wide SCSI disk drive at ID 0 of the wide bus, and non-wide devices at other IDs, enter this statement to enable wide SCSI operation for just that drive:

```
SCZDSP SCZR60/EW:0
```

If you have both wide and narrow SCSI devices attached to the wide bus, enable wide SCSI operation **only** for the wide devices. Use /EW without a device ID (to enable wide operation for the entire bus) only if all devices on the bus, both disk and tape drives, are wide SCSI devices.

Do not use the /EW switch with the narrow SCSI bus, even if you have wide SCSI devices attached to the bus using the appropriate adapters.

6.4^{oo}DEFINING DEVICES (DEVTBL)

Following the TRMDEF command lines is the DEVTBL command. It defines the devices your computer can access. You must always have at least one DEVTBL command line in the system initialization command file.



The computer already knows the System Disk, DSK0:, is present, so don't put DSK0: in the DEVTBL command line. If your computer has no devices other than DSK0:, enter DEVTBL alone on a line, and do not follow it with any arguments.

You can list sharable and non-sharable devices on separate DEVTBL lines, or you can list them together on the same DEVTBL line. A sharable device is one that all users can access simultaneously, such as a disk drive. A non-sharable device is one that only one user at a time can access, such as a magnetic tape unit. You **MUST**, however, separate the sharable devices from the non-sharable devices by inserting a slash (/) after the sharable devices, and before any non-sharable devices. For example, either of the following formats is correct:

```
DEVTBL SUB0 ,SUB1 ,SUB2
DEVTBL /STR0 ,MTU0
```

or

```
DEVTBL SUB0 ,SUB1 ,SUB2 ,/STR0 ,MTU0
```

If your computer has more devices than will fit on one DEVTBL command line, you can have as many DEVTBL command lines as you want, as long as they are not separated by other commands.

If you have a single physical disk divided into a large number of logical devices, you have two options to make your DEVTBL statements shorter. You can specify just the name of a single device with no unit number, and DEVTBL will read the hidden sector of the disk to determine the number of logical units to allocate. For example:

```
DEVTBL DSK
```

This format works only with SCSI drives.

Or, you can specify a range of logical unit numbers in a DEVTBL statement. For example:

```
DEVTBL DSK1-54
```

Be aware however, that if your system contains multiple disks of the same type, and they are all defined with the same device name, then you **must** use the dashed method of defining each drive's logical units. **AMOS will not search for additional drives with the same name unless they are defined with a DEVTBL statement.** For example:

If you have three identical system drives, and each drive has 10 logical units, then you can define each physical drive as follows:

```
DEVTBL DSK1-9
DEVTBL DSK10-19
DEVTBL DSK20-29
```

If you have identical SCSI drives divided into the same number of logical units, and they have consecutive SCSI IDs, you can identify all of them with a single DEVTBL statement. In this case, you could define three drives of ten logical units each with this statement:

```
DEVTBL DSK1-29
```



There are also important restrictions on how you specify DEVTBL command lines for devices controlled by certain disk controllers. See your *System Operator's Guide* before adding device definitions for hard disk devices.

As the computer processes the DEVTBL command line, it builds a device table in system memory. The file system consults this device table for device assignments. Here are some sample device names your computer might recognize:

DSK0	Logical unit zero of the System Device.
SUB5	Logical unit #5 of a hard disk subsystem drive.
MIN0	5.25" Floppy disk drive.
FLP1	3.5" Floppy disk drive.
STR0	Streaming tape unit.
MTU0	Magnetic tape unit.
TRM	The generalized terminal service driver. Allows file oriented input and output to any terminal connected to the computer.

After the computer is fully up and running, the DEVTBL command is a user command that tells you what devices are in the device table in system memory; it also tells you which devices are sharable among users. See your *System Commands Reference Manual* for information.



Remember, there must be a device driver in DSK0:[1,6] (or loaded into system or user memory), with the same name as the device listed on the DEVTBL line, and it must have a .DVR extension.

6.5^{oo}DEFINING DISK BITMAPS (BITMAP)

To write information to a disk, the AMOS file structure needs a disk allocation map (a bitmap). The BITMAP command sets up these maps. If the disks on your computer run under different disk controllers, then each type of disk must have its own device name and separate bitmap areas.



Floppy disk drives, which may use diskettes in several different formats, must have a different device defined for each type of format, even though the drives may run under

the same controller. Also, several types of hard drives can run under the control of the same Hard Disk Controller—but because these devices are different sizes, each type may have to have its own name and bitmap area. See your *System Operator's Guide* for information on hard disks.

The BITMAP command specifies the name of the device the bitmap is for. AMOS then sets up the bitmap for that device. There are three ways the BITMAP statement can appear, depending on what type of disk you have, and whether or not you are using paged bitmaps. Use the appropriate type of bitmap statement for your computer.

Non-self-configuring drives on computers without paged bitmaps require the three-letter name of the drive, the bitmap size, and the numbers of the logical units for that drive. For example:

```
BITMAP DSK,4311,0,1,2,3
```

On self-configuring drives AMOS can automatically read the bitmap size from the disk. If your disk has this capacity, you may leave out the bitmap size (but leave the comma) when you specify the line. For example:

```
BITMAP DSK,,0,1,2
```

If your computer supports "paged bitmaps," and you are using a SCSI disk, you only need to specify the three-character device name. A paged bitmap is one only partially in memory, thereby saving memory space. The portions of the bitmap needed at any given time are "paged" into memory from the disk. For example:

```
BITMAP DSK
```

Paged bitmaps are especially useful for large disks, but may be inefficient if used with small devices such as floppy disks. You may want to use paged bitmaps for your large drives, and traditional format bitmaps for smaller devices.

For information on the number of words needed for the bitmap of a particular floppy disk device, and information about hard disks, see your *System Operator's Guide*.

The monitor builds one sharable bitmap area in memory for each BITMAP command it finds. The BITMAP command also specifies the disks to be accessed by SYSTAT when SYSTAT prints the number of free blocks left on the devices.

6.6⁰⁰INCREASING THE SYSTEM QUEUE SIZE

The monitor has a general purpose queue system that several AMOS commands use, and which is also available to user programs. The queue contains a fixed number of longword blocks which are assigned and then returned during the course of processing. The number of queue blocks you need depends upon the size of your monitor and the tasks it performs.

You may find you need more queue space if your computer makes heavy use of the file locking system. The STAT program will show you the number of queue blocks available—if this number often drops below 20, you may want to allocate more blocks.

The monitor initially contains 80 queue blocks; you may add more by using the QUEUE command in the system initialization command file. Place the QUEUE command before any SYSTEM commands. The QUEUE command allocates additional queue blocks. For example, a QUEUE 20 line adds 20 blocks to the basic queue size of 80 to give a total queue size of 100 blocks.



If you're using a SCSI dispatcher, be sure to allocate a large number of queue blocks (especially if write-caching is enabled). Start with an initial allocation of 2000 additional blocks per hard drive, and adjust the allocation by observing the queue block usage over an extended period of time.

The command line for adding 2000 queue blocks is:

```
QUEUE 2000
```

For information on how an assembly language program can access the monitor queue, see your *Monitor Calls Manual*.

6.7^{oo}SETTING UP INTER-TASK COMMUNICATION (MSGINI)

The MSGINI command creates a memory area for the use of the Inter-Task Communication (ITC) system. This system makes it easier for devices and programs to communicate, and is necessary for certain programs, such as the Task Manager and the printer spooler. 16K is a recommended size. For example:

```
MSGINI 16K
```

6.8^{oo}SETTING UP THE ERSATZ INITIALIZATION FILE

An ersatz name is a name that substitutes for a device/account specification. It serves to make locations on your computer easier to enter and remember. For example, the System Operator's account, DSK0:[1,2], has the OPR: ersatz name. You can define ersatz names for your own accounts by listing them in an ersatz INI file. Then you need to add an ERSATZ command line in your system initialization command file to execute your ersatz INI file at boot-up, so the computer knows what ersatz names have been defined.

You can use the standard file name of ERSATZ.INI, or any unique name you choose. You can also define multiple ersatz INIs in your initialization file. For example:

```
ERSATZ ERSATZ.INI  
ERSATZ MYERZ.INI
```

There are two options available with the ERSATZ statement. The /B option lets you add blank entries to the ersatz definition list, so you can later use the ERSATZ command from command level to define additional ersatz names while the system is running.

Normally, if an ersatz definition file contains an ersatz name that has already been defined (in a previous file), the new definition is ignored. The /O option causes the definitions in this file to override any previously defined ersatz names.

Here is an example line using the two switches:

```
ERSATZ NEWERS.INI/B:10 /O
```

6.9^{oo}ADDING PROGRAMS TO SYSTEM MEMORY

You may put programs in the system monitor by using the SYSTEM command in the system initialization command file. These programs may be AMOS programs or your own programs. When the computer reads the system initialization command file, it loads into system memory the programs you've specified in the SYSTEM commands. These programs actually become part of the monitor, and so dynamically increase its size as they are loaded into memory.

The advantage of adding a program to system memory is it loads faster than programs that must be read from the disk.



The programs to be included in the monitor must be re-entrant and re-usable (that is, sharable by more than one user). If they are not re-entrant, there is a possibility of system failure when two users access the same program. Many of the AMOS programs are re-entrant. See your *System Commands Reference Manual* to see if a particular command program is re-entrant. If you load a program supplied by Alpha Micro that is not re-entrant into system memory, SYSTEM displays:

```
%Warning -- program is not re-entrant.
```

It is important to add the SYSMSG file to the system. This file (which will have a specific extension depending on the language it defines) defines the characters and conventions to be used by the current default system language. If this file is not loaded into system memory, all error and system messages will be printed as code numbers. It is a good idea to place this command first in the list of SYSTEM commands, so error-reporting is turned on as soon as possible. The file for American English is SYSMSG.USA.



In addition to defining SYSMSG.USA in the SYSTEM statements, it's a good idea to LOAD SYSMSG.USA early on in your initialization file. A good place to insert this line is immediately after the JOBALC statements. By loading this program into memory near the beginning of the boot file, any errors detected will be displayed as text, even before the SYSTEM statement is executed.

One of the common uses of SYSTEM in the system initialization command file is to include the AlphaBASIC runtime package (RUN.LIT or RUNP.LIT) in the monitor so each user does not need to load RUN into his or her own memory partition. You may also include the interactive compiler (BASIC.LIT) itself if you expect heavy development work by more than one user. If users on your computer will be making extensive use of the screen-oriented text editor VUE, you may want to load it into sharable memory by using the SYSTEM command.

It is important to load the device driver program of any peripheral devices you have connected to your computer into system memory, so programs can access that device. Peripheral devices include disk drives, floppy drives, backup devices, etc.

Another reason to use the SYSTEM command is if you want to include frequently-called user subroutines in the monitor. You can locate such subroutines by name from assembly language programs by using the SRCH and FETCH monitor calls. Again, if these programs are to be shared by several users, they MUST be re-entrant.

To include programs in the system monitor, use one SYSTEM command for each program. Follow each SYSTEM command with the file specification of the program you want to include. For example:

```
SYSTEM VUE.LIT
SYSTEM RUN.LIT
SYSTEM STR.DVR[1,6]
```

If your file specification does not include an extension, the monitor assumes a .LIT extension; if you don't supply an account specification, the monitor assumes account DSK0:[1,4].



You must place any SYSTEM commands after all other commands in the system initialization command file that expand the monitor size.

The SYSTEM command has another use beside the inclusion of programs in the system monitor. A SYSTEM command alone on a line in a system initialization command file (that is, not followed by a file specification), tells the system that monitor expansion is finished. The computer then flags the monitor as up and running. The computer also sets a flag in the system communication area indicating the system initialization is done except for final cleanup. Various commands (including SYSTEM) test this flag to see which mode to operate in.

For example, before the computer is up and running, the BITMAP command defines the disk bitmap areas in the monitor; after system initialization, the BITMAP command displays the memory locations of those bitmaps.

Whether or not you include any programs in the monitor, your system initialization command file MUST have a SYSTEM command without a file specification to tell the operating system the computer has been initialized. This blank SYSTEM command must be listed after any other commands that expand the monitor size (including any other SYSTEM commands).

After the computer is up and running, the SYSTEM command performs a new function as a user command telling you what programs are in system memory and the total size (in decimal bytes) of the monitor.

6.9.1[∞]Including Device Drivers in System Memory

All SCSI drivers for non-DSK devices, and all AM-219 device drivers, must be loaded in system memory.

There is one other situation in which you **MUST** include device drivers in system memory: If users on the computer will be accessing a device that is not the System Device from within AlphaBASIC or AlphaVUE, you must preload the driver for that device into system or user memory before using AlphaBASIC or AlphaVUE, or the device access will fail.

The reason for this is that to perform their special functions, AlphaBASIC and AlphaVUE have to handle their own memory setup—however, because of this, once AlphaBASIC or AlphaVUE is run, there is no room in memory to load a device driver.

For example, users often access the generalized terminal driver, TRM.DVR, within AlphaBASIC or AlphaVUE using statements similar to these:

```
OPEN #100, "TRM:TRM6", OUTPUT
```

or

```
> Unyank TRM:TRM6
```

If this is true on your computer, remember to keep TRM.DVR in system memory or preload it into user memory before using BASIC or VUE.



You may want to load the driver programs for all the devices you use on your computer into system memory, so they will always be available.

6.9.1.1[∞]Setting Up Write Buffering

Write buffering can increase system performance by making the process of writing data to the disk more efficient. When write buffering is on, the system stores data to be written in a buffer in memory and writes it to disk in larger segments rather than one block at a time. It waits to write until the disk is not busy, or until a time limit you set.

You can use write buffering on any AMOS system which uses an "enhanced" SCSI dispatcher (SCSI dispatchers are discussed earlier in this chapter). You set up write buffering by adding the /N option to the SYSTEM statement loading your disk device driver in system memory. The format is:

```
SYSTEM DVR:dev/N buffer-size flush-time
```

dev is the name of the disk device you want to set up write buffering for. **buffer-size** is the size of the write buffer; **flush-time** is the maximum length of time data can be in the write buffer before being written to disk.

For example:

```
SYSTEM DVR:SUB/N 200K 30
```

This sets up write buffering for all drives using the SUB driver. The write buffer can hold up to 200K of data, and, even if the disk is busy, any pending data will be written to disk after 30 seconds.



Even though you don't need to load the DSK driver into system memory, you can turn on write buffering for DSK devices by adding a SYSTEM statement for the DSK driver using the format above.

If your computer supports write buffering, you can find more information in your *Owner's Manual* or other system documentation.

6.9.2^oDefining Floppy Disk Drivers to Use Buffered I/O

The SYSTEM command can define floppy disk drivers to use buffered input/output. Doing so sets up a buffer pool for the device associated with the driver, and the number of physical disk accesses is greatly reduced, thereby increasing floppy transfer speed. In order to get a performance improvement, the driver must be loaded into system memory at boot time by placing a command with this format in the initialization file:

```
SYSTEM DVR:[driver-name].DVR/N {number-of-file-updates}
```

where **driver-name** is the three-letter device name given to the driver when it was configured, and **number-of-file-updates** is the number of file (and hence bitmap) updates that will take place each time before buffered bitmap and directory information is flushed to the floppy when "buffered writes" are active (see the MOUNT command in your *System Commands Reference Manual*). Leaving this option out causes the driver to default to "every" file update. Leaving the "/N" option off completely, or not loading the driver into system memory, causes the driver to perform traditional unbuffered input/output, providing no performance improvement. Here is an example:

```
SYSTEM DVR:MIN.DVR/N 20
```



This method of buffered floppy disk access does not work with drivers using the AM-219 controller.

6.9.3^oFront Panel Write Buffer Display

If your computer has a front panel with three bar graph LEDs, you can use the LEDs to display write buffer usage for up to three device drivers. This command is only useful for SCSI disks which have had write-caching enabled. To enable this feature, use this SYSTEM statement:

```
SYSTEM SCZWCD.SYS/N dev {dev} {dev}
```

dev is the name of the device driver you want to display buffer usage for. For example, this statement would display usage for all DSK devices on the top bar and SUB devices on the second bar:

```
SYSTEM SCZWCD.SYS/N DSK SUB
```

6.9.4^oEnabling Interpreted Prompts

When you use MUSER or SET to change the AMOS prompt from its default of a period, you can use codes to include variable information, such as the user name or logged in account, in the prompt. This is known as an interpreted prompt. For interpreted prompts to work, you must load the file PROMPT.SYS into system memory. Include this statement among the SYSTEM statements:

```
SYSTEM PROMPT.SYS/N
```

For more information on using interpreted prompts, see the MUSER or SET reference sheet in the *AMOS System Commands Reference Manual*.

CHAPTER 7

FINISHING THE SYSTEM INITIALIZATION

After the monitor processes the SYSTEM commands in the system initialization command file, the computer is technically up and running. There are a couple of things still left to do, however, before the initialization procedure is complete. You may now include any commands in the system initialization command file you want the monitor to perform automatically at the time of computer start-up.

These commands are all commands you can enter from the keyboard for yourself, but it is usually convenient to have the monitor perform them automatically every time you power up or reset the computer. For example, you can have the monitor mount the disks you are going to use, set up jobs, etc. We discuss some of these commands below.

After you have included the functions you want performed automatically, there is one last thing to do before system initialization is complete—de-allocate the temporary user memory partition in which the system initialization command file was processed. Use the MEMORY 0 command at the end of the system initialization command file to do this; this command also allows the Operator Job to gain access to all available memory not already assigned to other jobs the next time the Operator Job attempts to use a command.

7.1[∞]SETTING OPTIONS

The SET command can perform a variety of system functions. For example, a SET DSKERR command tells the computer to report any soft disk errors that occur. SET GUARD, as another example, guards a terminal from any messages or any FORCE'd input sent by other terminals.

Note the SET command only affects the job that used it. For example, the SET DSKERR command above only affects the job the computer comes up under. For information on forcing commands to other jobs, see below.

See the SET reference sheet in your *System Commands Reference Manual* for more information.

7.2[∞]SETTING UP JOBS

When the computer is reset or powered up, it automatically attaches the first job to the first terminal defined in your system initialization command file. Except for that special case, however, the computer does not automatically attach any jobs to terminals. If you want a job to be able to use a terminal for input and output, you must explicitly attach the job and the terminal. This process involves the use of such commands as ATTACH, KILL, FORCE, MEMORY, WAIT, etc. (explained below). The SETJOB command consolidates all these into one easy-to-understand line. The format of the line is:

```
SETJOB jobname,terminal-name,memory{,command{,command...}}
```

jobname is the name of the job you want to start, **terminal-name** is the terminal to ATTACH it to, **memory** is the amount of memory to assign to the job (you may specify K and M for kilobytes and megabytes), and **command** is an AMOS command, command file name, etc. For example:

```
SETJOB JONSON,EARVIN,150K,JONSON.CMD
```

JONSON.CMD might contain:

```
LOG 100,44
SET DSKERR
VER
LOAD MAGIC.PFK
RUN REMIND
```



SETJOB interprets any comma in the command part of the line as the beginning of a new AMOS command. This means you cannot refer to a command file in another account by including an AMOS account number. For example, the SETJOB line:

```
SETJOB JOB1 TRM1, 200K, JOB1.JIN[100,33]
```

will fail, because SETJOB will interpret the comma in [100,33] as the beginning of a new command. To avoid this situation, place any .JIN files to be run by SETJOB in SYS: and all command files in CMD:. Or, you can assign an ersatz name to the account containing the .JIN or command file and refer to the ersatz name in SETJOB.

7.2.1[∞]Attaching Terminals to Jobs

When the computer first begins to process the system initialization command file, it automatically attaches the first job listed in the JOBALC command and the terminal defined by the first TRMDEF command. Except for this special case, however, the computer does not automatically link jobs with terminals. (When a job is linked to a terminal, the job and terminal are "attached." When a job is not linked to a terminal, the job is "detached.")

A detached job must have a terminal attached to it before it can do terminal input or output. A detached terminal, on the other hand, can be accessed through terminal

service calls or the general TRM driver. You cannot attach a job to a detached terminal from that terminal itself; you must do it from another terminal.

To attach jobs and terminals, use the ATTACH command. Once a job is attached to a terminal, it uses that terminal for input and output. You can use the ATTACH command in several different ways:

```
ATTACH Terminal,Job
```

This command attaches the terminal and job named. If the terminal or job are already attached to other units, the ATTACH command detaches them before it attaches them to each other.

```
ATTACH Job
```

This ATTACH command attaches the user's own terminal to the job named (and detaches it from the current job).

```
ATTACH
```

This use of the ATTACH command lists the terminals currently attached, and the jobs to which they are attached. For more information, see the ATTACH reference sheet in your *System Commands Reference Manual*.



The SETJOB command performs an ATTACH command automatically. You do not need to use both commands in your system initialization file!

7.2.2^oInitializing Jobs (KILL)

To properly initialize the jobs on the computer, you should KILL the jobs you have defined. A KILL command sends a Control-C to the specified job; this puts the job at the monitor level, ready to receive and send data. The KILL commands must appear after any ATTACH commands, but before any FORCE commands are used to send commands or data to the job.

Do not kill the job the computer is coming up under (that is, the first job in the JOBALC command line).

Use one KILL command for each job on the computer (except the job the computer is coming up under). For example:

```
KILL OLIVER  
KILL STAN
```



The SETJOB command performs the KILL statement automatically.

7.2.3[∞]Forcing Input to Jobs

The FORCE command gives you a way of sending input to another job. To send one line of input to another job, use the FORCE command followed by the jobname and the input. For example:

```
FORCE SANDI LOG DSK2:22,2
```

The line above logs SANDI into the computer under account DSK2:[22,2]. You can also send several lines of input to a job by entering a carriage return after the jobname. For example:

```
FORCE SANDI
```

After that point, all lines of text that follow (up to a blank line) will be sent to the specified job (a blank line is a carriage return alone on a line). For example:

```
FORCE SANDI
LOGON                ; Log in SANDI
SANDI                 ; Input user name
RUN REMIND            ; Run a user program

; Blank line above ends FORCE
```



The SETJOB command performs the FORCE statement automatically, as well as any other AMOS commands you wish to include in your job initialization command file!

7.2.4[∞]Allocating Memory to Jobs

If you are going to allocate memory within the system initialization command file, use the FORCE and MEMORY commands. For example:

```
FORCE SANDI MEMORY 32K
```



Be careful not to use the MEMORY command to allocate memory to the job the system initialization command file is coming up under—this will cause the processing of the system initialization command file to stop.

Automatic re-assignment of all available memory only occurs after a job has been give ZERO bytes of memory. Be careful not to assign a job a small, non-zero amount of memory (such as MEMORY 1K), since that partition will be too small to load in and execute a program (such as MEMORY itself), and a larger partition will not be automatically assigned.



After FORCEing a MEMORY command to a job, use the WAIT command to give AMOS time to finish the memory allocation before going on to the next command in the system initialization command file. Specify the name of the job to whom you are FORCEing memory. For example:

```
FORCE OLIVER MEMORY 32K
WAIT OLIVER
```



Once again, SETJOB will do this too!

7.2.5°The VER Command

When your computer boots, all of the keyboards connected to active terminals are "locked," preventing any signals from those keyboards from interfering with the booting process. You need to use the VER command to unlock the keyboard of each terminal defined in your initialization file. See the example in Appendix A.

7.3°MOUNTING DISKS

The computer automatically mounts the System Disk (DSK0:) for you at the time of computer start-up. If you wish the computer to mount other disks as well, you have two options: You can include a MOUNT command for each disk logical unit, or you can specify the device name only (with a colon) and MOUNT will automatically mount all its logical units for you. For example, either of the following formats can be used:

```
MOUNT DSK1 :
MOUNT DSK2 :
MOUNT DSK3 :
MOUNT SUB0 :
MOUNT SUB1 :
```

or

```
MOUNT DSK :
MOUNT SUB :
```



The device-name-only format of MOUNT (as in the second example above) works only with SCSI drives.

You **MUST** mount a disk before you can read or write data to it.



Never mount a disk while another user is using it, or you might damage the bitmap.

7.4°SETTING UP EVENT LOGGING

Alpha Micro offers an event-logging utility program called LOGGER that runs on a background job and keeps a record of system events such as system errors. See your *System Operator's Guide* for more information. To run the event logger, set up a background job, and then FORCE that job to run LOGGER:

```
JOBALC LOGJOB ; Name the job
TRMDEF LOGTRM,PSEUDO,NULL,100,100,100 ; Set pseudo-terminal
;
ATTACH LOGTRM,LOGJOB ; Initialize the job
KILL LOGJOB
FORCE LOGJOB ; Force input into job
MEMORY 10K ; Give job 10K
LOG SYSTEM SERVICE ; Log job onto system
LOGGER 7 ; Execute LOGGER

; Blank line above ends FORCE
```

The background job that runs `LOGGER` needs at least 10K of memory.

The 7 in the above example sets the minimum severity an event needs to have to get logged. The higher the number, the more events are recorded in the log file. 7 is generally a good choice; higher settings can slow down system operation and take up disk space by recording too many events in the log file.

If you want the log to go to a file other than the default of `OPR:SYSLOG.SYS`, you can enter a file specification after the event level on the `LOGGER` command line.

7.5°°SETTING UP THE TASK MANAGER

The Task Manager allows you to run "batch" jobs on your computer. A task submitted to the Task Manager is run in the background by a special job. See your *Task Manager User's Manual* for more information.

7.6°°SETTING UP THE PRINT SPOOLER

A spooler is a program that sets up a queue (or waiting line) for a particular program. When a printer spooler is in control, requests for use of the printer are placed into a queue. As the printer becomes available, the spooler looks at the request at the top of the list, finds the file, and sends it to the printer. The spooler then removes that request from the queue. The printer prints files in the order of their requests in the queue.

The print spooler uses the Task Manager, and must be brought up by using the `TSKINI` program in the system initialization command file.

Setting up a printer spooler is a good example of the kinds of things you can ask the system initialization command file to do at the time of system start up. For a detailed explanation of how to set up the printer spooler, see your *System Operator's Guide*.

7.7°°SETTING UP ALPHACD

If you want an AlphaCD CD-ROM drive to be accessible to all users after booting, load ACD.DVR into memory using a SYSTEM statement. Then, if you want to mount all the CD logicals during bootup (be sure the CD is installed in the drive), enter this command after the last SYSTEM statement:

```
ACD
```

This makes all of the logical devices on the CD available. They do not have to be defined in a DEVTBL statement.

If you do not use the ACD command during bootup, you can mount the CD from AMOS command level by entering ACD at the AMOS prompt. To unmount the CD, enter ACD/U .

7.8°°FORCING THE OPERATOR JOB TO RUN TASKS

If you want your Operator Job to execute tasks or programs automatically when the computer boots, you can use FORCE on the Operator Job just before the MEMORY 0 command. When the MEMORY 0 command is executed, those FORCED commands will be run. For example:

```
FORCE JOB1
RUN REMINDS
WRITE
EDIT CALEND

; Blank line above ends FORCE
```

7.9°°MEMORY 0

The very last command in the system initialization command file MUST be a MEMORY 0 command. See the section above for a discussion of the MEMORY command.

CHAPTER 8

TESTING YOUR SYSTEM INITIALIZATION FILE

Once you have completed your initialization file, you need to test it to be sure it works correctly. This chapter explains the testing process, and shows you how to set up a minimal system initialization file.

8.1 THE MONTST

Log into the System Operator's account (OPR:). Type MONTST, followed by the name of your System Monitor file, and the initialization file you wish to test. For example:

```
MONTST AMOSL.MON,TEST.INI 
```

The computer will now boot under the TEST.INI file. If something should go wrong, press the RESET button on your computer and it will boot again using your regular initialization file. Once you are satisfied the test file is working correctly, you can use the RENAME command to rename it to the name of your system initialization command file. The MONTST command can also be used to test a new system monitor file, or both the monitor and the initialization files. For more information on MONTST, see your *System Commands Reference Manual*.

8.2 RENAMING YOUR TEST FILE

After you are sure your TEST.INI file works correctly, you can rename it to the name of your system initialization command file so you can boot from it when you push the reset button or when you power up the computer. But first, it is a good idea to make a copy of your initial system initialization command file under another name so you can refer to it in the future—just in case a minor problem crops up with your new system initialization command file, or you want to return to the earlier configuration. For example:

```
COPY AMOS32.OLD=AMOS32.INI   
RENAME/D AMOS32.INI=TEST.INI 
```

8.3 CREATING A MINIMAL FILE

There are times during addition of new hardware to the computer or when troubleshooting an existing computer when it is handy to create a minimal system initialization command file. This is a system initialization command file that contains the minimum number of commands under which your computer will boot.

The advantage of a minimal initialization file is it lets you track down the problem you are searching for or add a new device without worrying about interactions with extraneous jobs and programs.

Once the computer comes up with under the minimal initialization file and works well, you can then begin to add items from the original initialization file one by one, testing between each addition, until you are sure the computer is operating correctly.



If you are not familiar with the elements of a system initialization command file, make sure you have read the early parts of this manual before trying to build a minimal initialization file.

To create a minimal initialization file, make a copy of your system initialization command file. For example:

```
COPY MINI=AMOSL.INI
```

Now, edit the MINI.INI file with AlphaVUE, and reduce your computer to a one person, one terminal system.



NEVER modify the file itself—modify a copy.

You may want to eliminate any device definitions for non-System Device devices. Remember, the reason you are building the minimal initialization file will dictate exactly what items to remove. For example, if you are troubleshooting the printer spooler, you will probably want to leave the definitions for the printer and the spooler job.

Some items that **MUST** be in a minimal initialization file are:

- A JOBS command.
- At least one TRMDEF command (defining the Operator Terminal).
- A PARITY command.
- A JOBALC command.
- At least one DEVTBL command (even if with no argument).
- At least one BITMAP command.

- A MSGINI command.
- A SYSTEM SYSMSG.USA command (or other system message file).
- A final SYSTEM command with no arguments.
- A MEMORY 0 command at the end of the file.

A typical minimal initialization file might look like this:

```
:T
JOBS 1
TRMDEF CRT0 ,AM355=0 : 57600 ,ALPHA , 80 , 80 , 80
PARITY
JOBALC OPR
DEVTBL DSK1
BITMAP DSK
SYSTEM SYSMSG.USA
SYSTEM
MOUNT DSK1 :
MEMORY 0
```

APPENDIX A

SAMPLE SYSTEM INITIALIZATION COMMAND FILE

NOTE: The following sample INI file is based on an Eagle 300 system. Some line statements may not be applicable to *your* system configuration. The sample INI is only meant to show some of the various configuration methods, and a proper sequence of events. Be careful if using a SCSI dispatcher—you must use the correct version for your system, and you can only have one dispatcher enabled at any time. Other configuration options remain your choice, such as using the traditional print spooler versus using the task manager. For more details on setting up your initialization file, refer to the applicable sections of this manual, and the *System Operator's Guide*.

```
:T                                ; Enable command tracing
;
JOBS 11
;
JOBALC JOB1 ,JOB2 ,JOB3 ,JOB4 ,JOB5 ,JOB6 ,JOB7 ,JOB8 ,SPOOLA ,SPOOLB
;
QUEUE 2000
;
LOAD LOAD.LIT
LOAD DEL.LIT
LOAD SYMSG.USA                    ; Load system message interpreter
LOAD TRMDEF.LIT
;
TRMDEF TRM1 ,AM318=0:19200 ,AM62A ,150 ,150 ,150 ,EDITOR=15 ; Master terminal
;
XY=24                              ; Turn Graphics Off
;
VER                                ; Unlock keyboard
PARITY                             ; Clear memory
;
;                                ; ENABLE ONE SCSI DISPATCHER ONLY!
;SCZDSP SIMRR.SYS                 ; Eagle/Roadrunner simple SCSI dispatcher
SCZDSP SCZRR.SYS                 ; Eagle/Roadrunner enhanced SCSI dispatcher
;
;                                ; Define remaining AM-318 ports
```

```

TRMDEF TRM2,AM318=1:19200,AM62A,100,100,100,EDITOR=10
TRMDEF TRM3,AM318=2:19200,AM62A,100,100,100,EDITOR=10
TRMDEF TRM4,AM318=3:19200,AM62A,100,100,100,EDITOR=10
TRMDEF TRM5,AM318=4:19200,AM62A,100,100,100,EDITOR=10
TRMDEF TRM6,AM318=5:19200,AM62A,100,100,100,EDITOR=10
TRMDEF TRM7,AM318=6:19200,AM62A,100,100,100,EDITOR=10
TRMDEF TRM8,AM318=7:19200,AM62A,100,100,100,EDITOR=10
;
TRMDEF EPPA,PSEUDO,NULL,50,50,2      ; EPP0 parallel printer dummy
TRMDEF EPPB,PSEUDO,NULL,50,50,2      ; EPP1 parallel printer dummy
;
LOAD DEVTBL.LIT
DEVTBL DSK1-16                        ; Define 1st system drive
DEVTBL DSK17-33                       ; Define 2nd system drive
DEVTBL TRM,RES,MEM
DEVTBL MIN0,FLP1                      ; 5 1/4" and 3 1/2" floppies
DEVTBL MTX0                          ; Mag tape drive
DEVTBL /STR0                          ; Streaming tape drive
DEVTBL /EPP0,EPP1                    ; Parallel ports
DEL DEVTBL
;
LOAD BITMAP.LIT
BITMAP DSK                            ; Paged Bitmap for system drives
BITMAP MIN,100,0                      ; Standard Bitmaps for floppies
BITMAP FLP,180,1
DEL BITMAP
;
ERSATZ ERSATZ.INI                    ; Load ersatz name tables
ERSATZ MYERZ.INI
;
MSGINI 20K                            ; Assign some memory for messages
;
LOAD SYSTEM.LIT
SYSTEM SYMSG.USA                      ; Load system message interpreter
SYSTEM DCACHE.SYS/N/M/U 300K         ; Enable disk read caching
SYSTEM DVR:DSK/N 100K 60             ; Enable dispatcher disk write caching
SYSTEM CMDLIN.SYS
SYSTEM SCNWLD.SYS
SYSTEM TRM.DVR[1,6]                 ; Load device drivers
SYSTEM STR.DVR[1,6]
SYSTEM MIN.DVR[1,6]
SYSTEM FLP.DVR[1,6]
SYSTEM MTX.DVR[1,6]
SYSTEM ACD.DVR[1,6]
SYSTEM EPP.DVR[1,6]
SYSTEM                               ; End monitor expansion
DEL SYSTEM
;
SMEM.LIT 300K                        ; Shared memory pool
;

```

```
LOG OPR:
SYSTEM SERVICE
SET DSKERR
SET HEX
;
MOUNT DSK:                                ; Mount all DSK logicals
;
SETJOB JOB2,TRM2,256K,JOBSET.INI          ; Attach additional jobs
SETJOB JOB3,TRM3,256K,JOBSET.INI
SETJOB JOB4,TRM4,256K,JOBSET.INI
SETJOB JOB5,TRM5,256K,JOBSET.INI
SETJOB JOB6,TRM6,256K,JOBSET.INI
SETJOB JOB7,TRM7,256K,JOBSET.INI
SETJOB JOB8,TRM8,256K,JOBSET.INI
;
ATTACH EPPA,SPOOLA                        ; Enable EPP0 parallel printer
KILL SPOOLA
FORCE SPOOLA
MEMORY 10K
LOG SYS:
SYSTEM SERVICE
LPTINI EPP0.INI

WAIT SPOOLA
;
ATTACH EPPB,SPOOLB                        ; Enable EPP1 parallel printer
KILL SPOOLB
FORCE SPOOLB
MEMORY 10K
LOG SYS:
SYSTEM SERVICE
LPTINI EPP1.INI

WAIT SPOOLB
;
CACHE                                      ; Lock some files in cache memory
OFF
ON
LOCK/MFD DSK0:
LOCK/UFM DSK0:[1,4]
LOCK/FILE VUE.LIT,COPY.LIT,DIR.LIT
EXIT
;
DEL *
;
MEMORY 0
```

APPENDIX B

GLOSSARY

- AMOS COMMAND LEVEL** When you are at AMOS command level, you are communicating directly with AMOS (the Alpha Micro Operating System) and not with a program AMOS is executing.
- AMOS PROMPT** When you're at AMOS command level, you see the AMOS prompt symbol, which tells you the operating system is ready for you to enter a command. This prompt may be the system default, a period (.), or it may be defined using SET.
- BASIC** Beginners All-purpose Symbolic Instruction Code. A simple, easy to use programming language.
- BINARY** Data in the form of a series of 0's and 1's. This is how data is represented at the core level of the computer.
- COMMAND LINE** Whenever you enter a command to AMOS, you include the name of the command optionally followed by device and/or file specifications, option switches, etc. The entire input line is called a command line.
- COMMAND FILE** A command file is an ASCII text file containing valid AMOS system commands and file specifications. It can contain most commands and data you can enter at AMOS command level (including the name of another command file). As AMOS processes a command file, it performs the functions called for by each line of the file. Command files can also contain several special symbols that affect the way the file is displayed on the terminal screen as it is processed, and that allow the file to ask for input from the user. You may also specify text arguments which AMOS substitutes in place of special parameter symbols.

CONTROL SEQUENCE

A control sequence is when you use the CONTROL key (sometimes labeled CTRL) in combination with other keys to affect what is occurring on your screen or during the execution of a command or program.

To use a control sequence, hold down the `CTRL` key and press another key. One of the most common control sequences is Control-C, which is used to interrupt commands and programs. To do this, hold down `CTRL` and press `C`.

Throughout this manual, we represent a control sequence by showing the keys you press, like this:

press `CTRL` / `C`

DEFAULT

When you leave information out of a command line, AMOS often has a set of information it substitutes for the missing items. For example, if you don't tell AMOS what account a file is in, it usually assumes it is in the account you are currently logged into.

Defaults vary among commands. Check the reference sheet for a specific command to see what defaults it uses. In particular, the special commands called wildcard file commands handle defaults differently than other commands on the system.

DELIMITER

A character marking a division between two pieces of data. For example, in the date display 12/04/86, the slash "/" is a delimiter. A dash "-" is also a common delimiter.

DISK DRIVE CONTROLLER

A board or paddle-board containing the hardware interface between the physical disk device and the Mass Storage Bus controller (for example, the AM-515). The disk drive controller is specific to the type of disk it controls. In this case, the disk drive controller is a Xebec, and is "between" the AM-515 and the physical disk drive.

DISK FILE

An area of memory on a hard disk containing a program, text, or data file. Unless such a file is erased, it resides permanently on the disk.

DSK0:

The first logical device of a system (even if the entire physical unit is configured as one logical device) is known as the System Disk, or DSK0:.

ECHO	A response by your terminal to what you press on the keyboard. If you press a key (say the "a"), and you see a response on your terminal screen (an "a" appears), then your terminal is in ECHO mode (the normal way for it to be). Sometimes—for instance, when you are entering a password, the terminal echoing can be turned off, so what you type goes to the computer, but is not displayed on the screen.
FILE LOCKING	A process by which a file being used by one program is "locked," so any other program trying to access it will be told the file is in use. This prevents problems occurring from two programs working on the same file at the same time.
FILE SPECIFICATION	Data on a disk is organized into logically-related groups called files. Whenever you want to identify a file to an AMOS command, enter that file's specification, which includes its name, and where it is located, if necessary.
INITIALIZATION	The process of "starting up" something, whether it is the computer system itself, or a part of the system, such as a print spooler or AlphaVUE. The initialization allows you to customize the thing being initialized to your particular needs.
INPUT	Any data coming into the program or computer. Input may come from the user at the keyboard, from a data file, or from some other external device.
JOBS	The system initialization command file defines to AMOS a "job" for each terminal connected to the computer. AMOS then knows the characteristics to be used with that terminal—how much memory to give it, etc.
LOGICAL DEVICE	A logical device is an abstraction. The computer cannot place or access groups of data on a disk correctly unless it has a name for the disk, and master and file directories of the contents of the disk available to it. So while the physical disk may be a Winchester technology 400 Mb hard disk to you, it is PLD0:, PLD1:, and PLD2: or the like to the computer. It is important to remember the logical devices a disk system is configured to have are only related to the physical disk drive. Exactly how the logical devices are configured is up to you, within certain physical limitations.

MOUNT	The process of preparing a disk device for use.
NON-SELF-CONFIGURING DISK	A disk drive that must have a specific disk driver program made for it so it can communicate with AMOS.
PHYSICAL ADDRESS	A Winchester controller board has a certain number of fixed addresses, or ports, to which Winchester technology disk devices may be physically attached by cables. When the system boots, AMOS tells the controller which disk drive is associated with address 0, which is associated with address 1, and so on. Then the devices are given names, and similar devices are given numbers.
PHYSICAL UNIT/DEVICE	A physical disk device includes one or more magnetically sensitive disks, and read/write heads to record and read back data recorded in tracks on the disk(s).
QUEUE	A "waiting list". For example, when printing files, a queue is used so the files sent to the printer do not print all at the same time, but rather in an orderly way—first come, first served.
RE-ENTRANT	A program is <i>re-entrant</i> when it can be used by more than one user at a time, and is therefore able to be loaded into System Memory, where it can be shared by all users.
RE-USABLE	A program is <i>re-usable</i> when it can be interrupted during its operation, and then resumed again, or when it can be run again after it has run. Since AMOS is a multi-user system, most of its programs are <i>re-usable</i> . Re-usable programs cannot be loaded into System Memory (unless, of course, they are also Re-entrant).
SAVE	The process of transferring a file from user memory (which is temporary) to a disk (which is permanent).
SELF-CONFIGURING DISK	A disk drive that uses a "generic" disk driver program rather than a specific disk driver. For example, a 400 Mb disk and a 80 Mb disk could use the same generic driver program if they are both self-configuring disks.

- SEQUENTIAL** A type of file storage in which each item of data follows the previous item of data in order in the storage on the disk. Sequential data files are slower for data retrieval than random files, but are easier to access and work with.
- SPOOLING** A method of sending a file to a printer queue, so it can await printing.
- SWITCH or OPTION** Many AMOS commands and programs allow you to select among several options by including switches on a command line. A switch is a slash (/) followed by one or more characters. You can sometimes include several switches on one command line.
- The specific form switches take varies depending on the particular command. Some commands expect every single character after a slash to represent a different switch. For example:
- MAP/FSR** (RETURN)
- Others require each switch begin with a new slash. For example:
- PRINT NET.BAS/COPIES:2/BAN/HE** (RETURN)
- See the reference sheet in your *System Commands Reference Manual* for a particular command to see the switches for it.
- SYSTEM MEMORY** An area of memory available to all of the users of the computer. Any file loaded into this memory area (by use of the SYSTEM command within your system initialization file) can be used by any person or program. The advantage of using system memory is that, if a file is used regularly by many users, each user does not have to load a copy of that file into his or her own user memory. This saves both time and memory space.
- USER MEMORY** An area of memory available to one user. You can load files into user memory using the LOAD command, and remove files by using the DEL command.
- USER NAME** A name given to a user, allowing the user to easily log in to a specific account, and identifying that user's access level and priveleges.

WILDCARD

A symbol that can represent a range of other characters. You might want to think of it as a joker in a deck of cards. For example, if you wanted to erase three files: FILE01.DAT, FILE02.DAT, and FILE03.DAT, you could type all this:

```
ERASE FILE01.DAT 
ERASE FILE02.DAT 
ERASE FILE03.DAT 
```

or you could type:

```
ERASE FILE0?.DAT 
```

where the question mark symbol "?" represents all characters, or:

```
ERASE *.DAT 
```

where the asterisk represents all combinations of characters (in this case, all filenames). Of course, you would not use the second command if you had other .DAT files you wanted to keep.

**WILDCARD FILE
COMMAND
SWITCHES**

Wildcard file commands distinguish between two types of switches: *file switches* and *operation switches*. If a file switch is directly after a file specification, it affects only that file. For example:

```
ERASE MTDVR.M68 , MTDVR.LIT / QUERY , MTDVR.OBJ 
```

tells ERASE to ask for confirmation before erasing MTDVR.LIT. It erases the other two files without asking for confirmation.

An operation switch affects all files on the command line, no matter where it is placed. For example, the /WIDE option with the DIR command affects the directory display for all specified files, no matter where it appears on the command line.

Wildcard file commands allow you to set the default switch by placing the switch in front of a file specification. For example:

```
ERASE / Q MTDVR , MTDVR.OBJ / NO Q , SRCFIL.BAS 
```

tells ERASE to ask for confirmation before erasing the first and third files specified on the command line.

See your *AMOS User's Guide* for more information on wildcard file command switches and default switches.

Document History

Revision 00 - AMOS Release 2.0 - (Printed 3/88) - New Document

Adapted and separated from the *AMOS System Operator's Guide*.

Revision 01 - AMOS Release 2.2 - (Printed 4/91)

Added information on separating the .INI file into more than two files to Chapter 3. Added a note to Chapter 4 about a JOB for BACKUP. Added information about terminal drivers to Chapter 5. Added information about shared memory and floppy disk buffered I/O to Chapter 6.

Revision 02 - AMOS Release 2.3 - (Printed 9/96)

Modified Chapters 5, 6 and 7; added SCSI dispatcher, AlphaCD, and interpreted prompts; changed SETJOB format and TRMDEF #tcb-number; other small changes. Updated sample in Appendix A.

Revision 03 - AMOS Release 2.3A - (Printed 5/97)

Removed specific IDV information, added disabling Super I/O, and made small correction to line editor section in Chapter 5. Updated SCSI dispatcher section and added write buffering in Chapter 6.

INDEX

.IDV	5-2
:T	4-1
Abbreviations	1-3 to 1-4
Account number	1-3
ACD command	7-7
Allocating memory to jobs	7-4
Alpha Micro operating system	1-1
ALPHA.TDV	5-4
AlphaCD	7-7
Alternate drivers	5-6
AMOS command level	B-1
AMOS prompt	1-3, B-1
ATTACH	7-2
Attaching terminal to jobs	7-2
BASIC	B-1
Batch tasks	7-6
Binary	B-1
BITMAP	6-5
Bootup	1-2
Braces { }	1-4
Buffered I/O	6-11
Caret ^	1-4
CD-ROM setup	7-7
Circumflex ^	1-4
Command file	1-1, B-1
Definition	1-1
Command line	B-1
Control sequence	B-2
Control-characters	
CTRL/ C	1-4
Default	B-2
Defining jobs	4-1
Defining terminals	5-1
Delimiter	B-2

Device drivers	6-10
Device independence	1-1
Device-name	1-3
Devn:	1-3
DEVTBL	6-4
Disabling Super I/O	5-3
Disk drive controller	B-2
Disk file	B-2
Dispatcher, SCSI	6-2
DSK0:	B-2
Echo	B-3
ERSATZ	6-7
Ersatz initialization file	6-7
ETHZON	5-7
Event Logging	7-5
External ports	5-2
File locking	B-3
File specification	1-3, B-3
Filespec	1-3
Floppy disk drivers	6-11
FORCE	7-4
Front panel write buffer display	6-12
Graphics conventions	1-3
I/O port address	5-2
Initialization	B-3
Initializing jobs	7-3
Input	B-3
Input/output	
Buffered	6-11
Input/output redirection	5-5
Inter-task communication	6-7
Interface drivers	5-2
PSEUDO	5-3
Super I/O	5-3
Internal ports	5-2
Interpreted prompts	6-12
ITC	6-7
JCB	4-2
Job	4-1
Job control blocks	4-2
JOBALC	4-2
JOBS	4-1, B-3
Keyboards	
Unlocking	7-5

Keycap symbol	1-3
KILL	7-3
LOGGER	7-5
Logical device	B-3
MEMORY	7-4
MEMORY 0	1-2, 7-7
Minimal initialization file	8-2
Monitor	1-1
MONTST	3-1, 8-1
MOUNT	7-5, B-4
Mounting disks	7-5
MSGINI	6-7
Non-self-configuring disks	B-4
Non-sharable devices	6-4
NULL driver	5-4
Operator job	1-2, 4-1
Operator terminal	1-2
Option	B-5
Paged bitmaps	6-6
PARITY	6-1
Physical address	B-4
Physical unit	B-4
Print queue	7-6
Print spooler	7-6
Prompt	
interpreted	6-12
PROMPT.SYS	6-12
PSEUDO driver	5-4
QUEUE	6-6, B-4
Re-entrant	6-8, B-4
Re-usable	6-8, B-4
Redirection	5-5
SAVE	B-4
SCSI dispatcher	6-2
and queue size	6-7
SCZ190	6-2
SCZDSP command	6-2
SCZR60	6-2
SCZRR	6-2
SCZWCD	6-12
Segmenting the .INI file	3-1
Self-configuring disk	B-4

Sequential	B-5
SET	7-1
SET TERMINAL DRIVER	5-6
SETJOB	7-2
Setting up jobs	7-2
Sharable devices	6-4
Shared memory	6-1
SIM190	6-2
SIMR60	6-2
SIMRR	6-2
Slash /	1-4
SMEM	6-1
Spooling	B-5
Super I/O	5-3
Switch	B-5
Symbols	1-3 to 1-4
SYSMSG	6-8
SYSMSG.USA	6-8
SYSTEM	6-8
System initialization command file	1-1
Definition	1-1
Function	2-1
Sample	A-1
Testing	3-1, 8-1
System memory	6-8, B-5
Device drivers	6-10
Floppy disk drivers	6-11
System queue size	6-6
Task Manager	7-6
TCBs	5-6
TDVDEF	5-6
Terminal control blocks	5-6
Terminal drivers>Building your own	5-6
Terminal drivers>Defining alternate drivers	5-6
Terminal interface boards	5-2
Testing your system initialization file	8-1
TRMDEF	5-1, 5-6
ALPHA driver	5-4
EDITOR	5-5
In-buffer	5-5
In-width	5-5
Interface	5-2
Modem-driver	5-6
NULL driver	5-4
Out-buffer	5-5
PSEUDO driver	5-4
Terminal	5-4
Terminal name	5-2
TSKINI	7-6

Unlocking keyboards	7-5
User memory	B-5
User name	B-5
VER	7-5
VTSER	5-6
WAIT	7-4
Wildcard	B-6
Wildcard file command switches	B-6
Write buffer display	6-12
Write buffering	6-10