
Apple III
Standard Device
Drivers Manual
apple III
Apple III Diskware Program
System Utilities

Notice

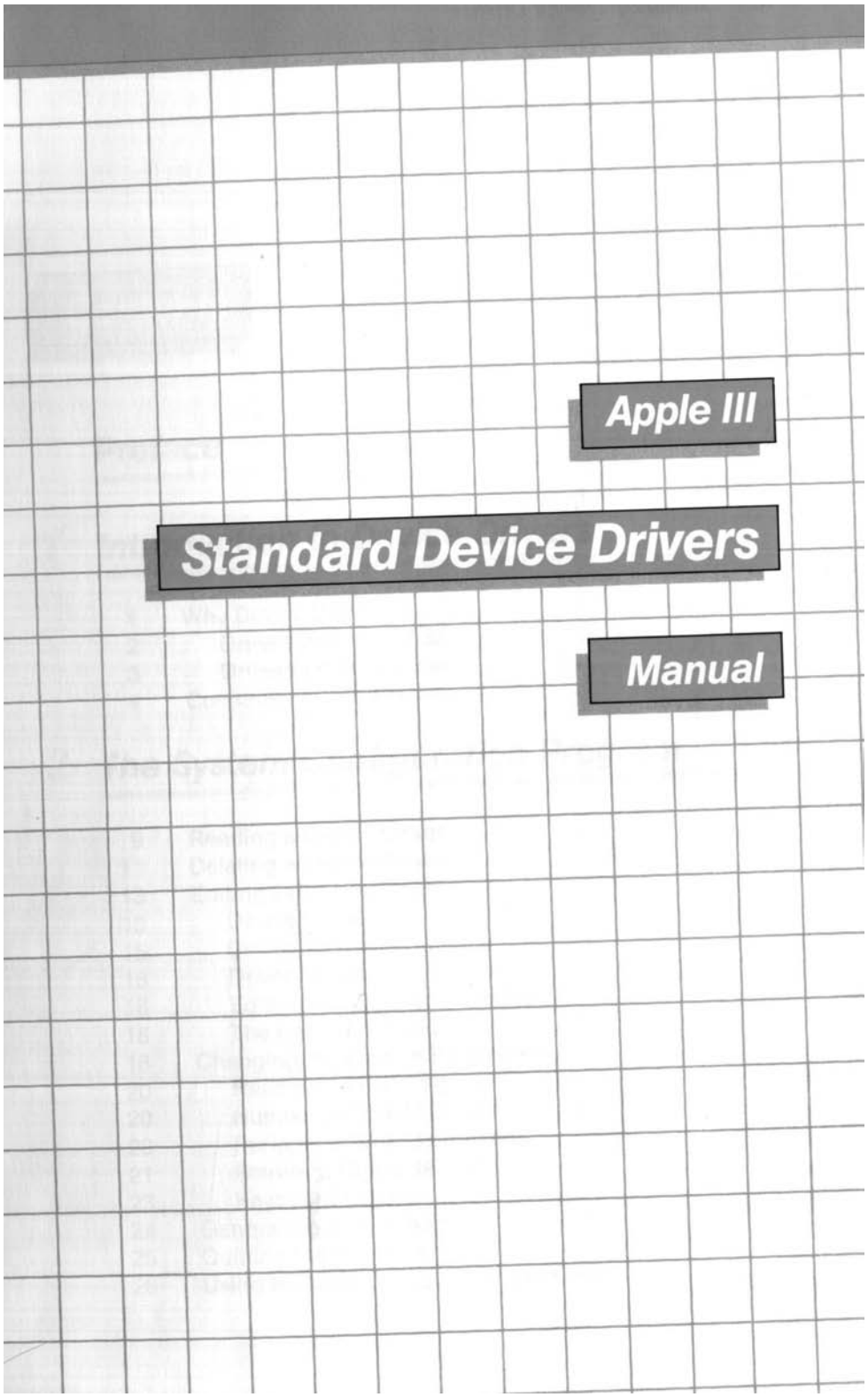
Apple Computer reserves the right to make improvements in the product described in this manual at any time and without notice.

Disclaimer of All Warranties And Liabilities

Apple Computer makes no warranties, either express or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Apple Computer software is sold or licensed "as is." The entire risk as to its quality and performance is with the buyer. Should the programs prove defective following their purchase, the buyer (and not Apple Computer, its distributor, or its retailer) assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Apple Computer be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software, even if Apple Computer has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

© 1981 by Apple Computer
10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010

Apple Product # A3L0008B



Contents

Appendices

Preface

This manual describes the standard device drivers that control the input/output devices built into your Apple III and some of the more commonly attached devices. This manual will be useful if you are writing programs that use these devices. The information about the .PRINTER device driver will also be useful if you are using a printer other than a Qume printer with your Apple III.

When you add a peripheral input/output device to your Apple III, you will need to use the System Configuration Program, which is supplied on the Utilities diskette. This manual tells you how to use the System Configuration Program to install or remove device drivers and to modify the parameters in device drivers; you'll need to do this whenever you install or remove a peripheral device.

If you have not already done so, you should read the Apple III Owner's Guide. It gives an introduction to the use of the System Configuration Program and the Utilities Filer. It also describes the way the operating system in the Apple III uses files and devices. Once you know how your Apple III uses devices in general, you can read this manual to learn how to use the specific devices it describes.

The device drives described in this manual are:

- the Console driver, which controls the keyboard and the text display;
- the Graphics driver, which controls the different modes of graphics displays;

- the Printer driver, which controls the built-in RS-232-C serial port for output only; for example, a printer;
- the RS-232 driver, which controls bidirectional operation of the built-in RS-232-C serial port; and
- the Audio driver, which controls the built-in speaker.

If you have purchased other peripheral devices or device drivers, you should refer to the manuals accompanying those products for information about their device drivers.

1

Introduction to Device Drivers

Why Device Drivers?

The Apple III's Sophisticated Operating System (SOS) uses special programs called *device drivers* to communicate with all peripheral devices. Whether a given device is built in or added later, SOS uses a device driver to exchange information with it. This manual describes the device drivers (the console, graphics, printer, RS-232, and audio drivers) that are supplied with every Apple III computer system. It also describes the operation of the System Configuration Program, which is used to install and modify these and other device drivers.

Peripheral devices such as the keyboard, video screen, speaker, and communications ports are the eyes and ears and other senses of your computer. Device drivers connect these senses to SOS. A device driver performs four necessary functions:

- It processes data generated by your programs and sends it to the device as output.
- It processes data generated by the device and sends it to your programs as input.
- It enables your programs and SOS to control the operation of the device and of the driver itself.

- It enables your programs and SOS to read the status of the device and of the driver itself.

Not all drivers perform all of these functions. Some drivers can process data in only one direction, such as the printer driver, which can only process output.

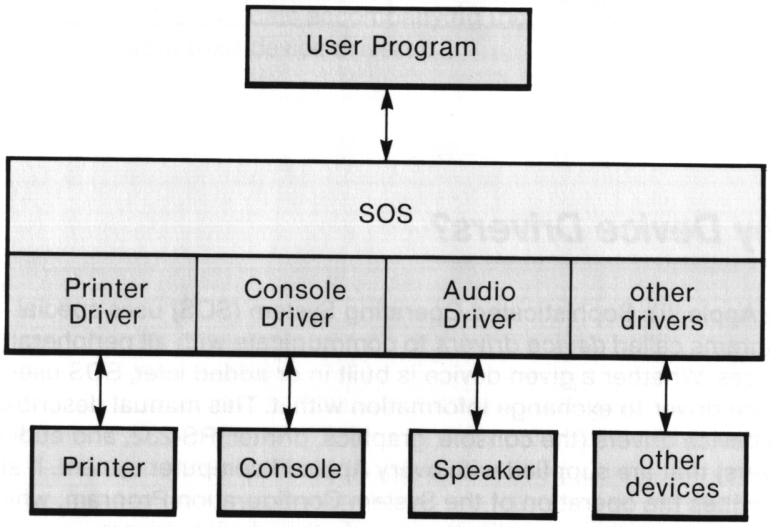


Figure 1: Drivers are part of SOS

Drivers are Part of SOS

When you boot the Apple III, the operating system is loaded into memory. When the operating system starts working, one of the first things it does is to fetch the device drivers from the boot diskette and store them in the Apple III's memory. Without these drivers, SOS is effectively blind, deaf, and mute.

Drivers are stored in a file named SOS.DRIVER on each SOS boot diskette. A file with this name must be present for SOS to complete the bootstrap operation. The information in this file is called the *System Configuration*. You can examine and modify this information with the System Configuration Program, described in the next section.

Along with SOS, the device drivers stay in the Apple's main memory until you turn off the power or re-boot the system. All communications between your programs and the device drivers are handled by SOS.

Once the drivers are loaded in the Apple III's memory, they can't be changed without re-booting the system (with a few exceptions, such as parameters in the RS232 driver). If you want to add a new device driver, remove a driver you don't need, or change a driver's default parameters, you must use the System Configuration Program to make a new SOS.DRIVER file, and then re-boot.

Devices Look Like Files

Even though the Apple III uses two kinds of peripheral devices, block and character, the data files your programs send to and from these two kinds of devices look just alike. Device drivers for character devices make their devices look like data files, which your program can read and write with normal file operations. For further discussion of files, see the *Apple III Owner's Guide*.

For example, a BASIC program that lists itself on the printer could be written like this:

```
10 OPEN#1, ".PRINTER"
20 OUTPUT#1
30 LIST
40 CLOSE#1
```

and a Pascal procedure to print a message on the printer might be written like this:

```
procedure print_msg(msg_text:string);
  var f:text;
  begin
    rewrite(f,'PRINTER');
    writeln(f,msg_text);
    close(f);
  end;
```

Device files are character files. You can open, read, write, and close device files, but you cannot read from them or write to them as random-addressed records, and you can't create, delete, or rename them.

Communications Formats

The basic unit of communication between your programs and a device driver is the *byte*. A byte consists of eight bits, so it can have any of 256 distinct values. Data passes between programs and drivers as streams of bytes. The byte streams can be any length, but the bytes themselves are usually treated as independent units.

Each byte that is sent to or received from a device driver can be interpreted as one of four things: a text character, a command, an argument of a command, or a byte of binary data. Drivers and programs determine whether a given byte is a text character, a command, an argument, or data by the value of the byte and the context in which the byte appears.

Text is usually printed information that a person will type or read. All of the devices that you are likely to use with your Apple III transmit and receive text using the American Standard Code for Information Interchange (ASCII). This code defines a set of 128 characters with values from Ø to 127. Each of these characters can be stored in one byte.

Commands are used to tell device drivers to perform certain functions and to find out when something has happened to the device or the driver. A command is a byte with a value from Ø to 31 embedded in the data stream sent to the driver. The ASCII defines these values as control characters; they are not used for text characters. Command bytes are also called control codes.

Arguments are additional data that some commands need in order to function properly. An argument byte is one of a fixed number of bytes following a command byte; the number of argument bytes is a characteristic of each command. Even though a byte has the same value as a command byte or a text character, it will be interpreted as an argument if it follows a command that requires arguments.

Data bytes can have any value from Ø to 255, and, like arguments, are identified by means of a byte count. If a driver is transferring program code or graphics data, for example, it would first transmit an argument telling the receiving device exactly how many data bytes to expect.

2

The System Configuration Program

The System Configuration Program is a tool for adding, removing, and modifying device drivers so as to change the configuration of the operating system. You will need to do this any time you add or remove a peripheral device such as a disk drive or a printer.

The *System Configuration* is stored in the file SOS.DRIVER on the boot diskette. It determines how SOS communicates with the peripheral and mass storage devices built into or attached to your Apple III. In addition to the device drivers themselves, the system configuration includes four *system parameters*:

- the number of disk III drives you have connected to your Apple III;
- the character set your Apple III uses to display text;
- the ASCII character codes assigned to each key on the keyboard; and
- the location of the peripheral interface cards in the four peripheral connectors inside the Apple III itself.

To add, remove, or change a device driver or system parameter, you use the System Configuration Program to edit a SOS.DRIVER file, adding drivers from other files as appropriate. You then store this new version of SOS.DRIVER on a SOS boot diskette and use the diskette to boot your Apple III.

Once you have made your new configuration and verified that it works properly, you should put a copy of the new SOS.DRIVER file into the root directories of the appropriate boot diskettes.

Every boot diskette may contain a different system configuration in the file SOS.DRIVER. Although the system parameters for your Apple III will probably be the same in each one, the drivers are likely to be different. Only those drivers that are in the system configuration you boot are available for use, until you re-boot the system with a different disk. On the other hand, the more drivers there are in a given configuration, the more disk space and memory the drivers will occupy. On each diskette supplied by Apple, there is a different set of drivers, appropriate to your use of the programs on each diskette. If you don't need some of these drivers, you may delete them to save space. Likewise, if you need a driver, add it to the system configuration on the appropriate boot diskette.

You must make sure that each boot diskette you create contains the appropriate system configuration. In particular, any system configuration you make must support the .CONSOLE device driver in order for your Apple III to operate.

The System Configuration Program is just a special-purpose editor you use to create and modify SOS.DRIVER files. The System Configuration Program is supplied on the Apple III Utilities diskette.

Boot the Utilities diskette using the standard procedure for booting diskettes, described in the *Apple III Owner's Guide*. The screen will show the Utilities menu. Select the SYSTEM CONFIGURATION PROGRAM option (refer to the *Apple III Owner's Guide* for a full description of option selection and other features of the Utilites Diskette). The screen will show the Configuration menu:

Photo 1. Configuration Menu

This menu shows the SCP options. You can read in, remove, and edit device drivers, change the system parameters, and generate a new configuration file. After you finish using an option, the screen will show the Configuration menu again.

Whenever you select an option from the Configuration menu, the diskette with the System Configuration Program must be in the drive it was in when you started using the program. This is normally the Utilities diskette in the built-in disk drive. If you have removed this diskette when you select an option from the configuration menu, the system will ask you to put it back.

You start creating a system configuration by reading in an existing configuration and editing it with the SCP. As you add device drivers and change system parameters, the program keeps this data in the Apple III's memory. When you are satisfied with the configuration you have made, the program writes all the data into a file on a diskette.

You should start by reading your current system configuration into memory. This will give you the standard device drivers and system parameters, which you can change as you wish.

An outline of a normal session to edit the system configuration on one boot diskette is shown below. The following sections of the manual explain each option in detail.

0. To protect yourself from accidental destruction of your valuable boot diskettes, start with a copy of the boot diskette you want to re-configure. To make the copy, use the Utilities Filer as described in the *Apple III Owner's Guide*.

1. Boot the Utilities diskette and select the SYSTEM CONFIGURATION PROGRAM option.

2. Select the option READ A DRIVER FILE and get the current SOS.DRIVER file from the boot diskette.

3. Select the option DELETE A DRIVER FILE to remove any device drivers you no longer need.

4. Select the READ option again to load new drivers from other diskettes, if necessary.

5. Select the option EDIT DRIVER PARAMETERS to configure each driver to suit your needs. Instructions on special configurations for a given driver are included in the documentation of that driver.

6. Select the option CHANGE SYSTEM PARAMETERS to set the number of disk drives, the character set, the keyboard layout, and the location and function of peripheral interface cards in the Apple III's four peripheral connectors.

7. Select the option GENERATE NEW SYSTEM to verify that the changes you have made are valid and to write the new configuration onto the copy boot diskette. You may write it to a file with any name you wish, but typically it will be SOS.DRIVER. If a file with the name you select already exists, the program will give you the option of unlocking and removing the previous file before writing the new one. You will need to do this if there isn't enough room on the boot diskette for both the old file and the new one.

8. Select the QUIT option to leave the System Configuration Program, then use the newly configured diskette to boot the system.

9. Verify that each driver in the new configuration works properly. If the new configuration does not work, re-boot the Utilities diskette and use the System Configuration Program to read the faulty configuration and repair it.

Reading a Device Driver

The READ A DRIVER FILE option enables you to read device drivers and add them to the configuration being built in the Apple III's memory. When you select this option, the screen shows the following:

Photo 2. Read a Driver File

The top section of the screen contains a list of the names of all device drivers currently in memory. Initially this list is empty; as you read in drivers, it will grow. The System Configuration Program can keep track of up to 32 drivers at once, depending on the amount of memory each driver requires.

To read a driver, insert the diskette containing the driver file and specify the pathname of the file that contains the driver. The System Configuration Program gives you three different ways to specify the file you want:

- Type the complete pathname of the file and press RETURN;
- Use the right and left arrow keys to edit the default pathname that appears on the screen; or
- Type a file pattern consisting of an incomplete pathname with the wildcard (=) in it, press down-arrow, then select a pathname from the menu of names that appears.

The editing and menu-selection methods used here are the same as the file-selection methods described in the chapter THE OPERATING SYSTEM AND FILES in the *Apple III Owner's Guide*. You can use the READ option to load either of the two kinds of driver files. One kind of driver file, sometimes called a driver code file, contains only a device driver or module in code form. A separate device driver supplied with a peripheral is usually this code-only form of driver file. The second kind of driver file contains system parameters in addition to driver code. This kind of driver file is created with the System Configuration Program; examples include SOS.DRIVER and CONSOLE.DRIVER.

When you create a new configuration, you use the READ option to load driver code files and driver files that include system parameters. The system parameters are automatically loaded with the first driver file you read that has parameters in it.

To load the current system configuration from one of your boot diskettes, insert the diskette into a disk drive and type the pathname of the configuration file. For example, if you use the built-in drive, type:

```
.D1/SOS.DRIVER
```

and press RETURN.

When the file has been read, the names of the device drivers the file contains will be added to the list at the top of the screen. Because this file was generated by the System Configuration Program and so contains system parameter data, and no system parameter data has yet been set, the message

System Parameters are being loaded from the file.

appears briefly at the bottom of the screen.

A group of one or more consecutive driver names each preceded by a plus sign (+) indicates that they are part of a *driver module*. A driver module is a group of related device drivers that share programming or resources. They are grouped this way for more efficient use of the Apple III's time and memory. All consecutive drivers marked with plus signs belong to the same module as the preceding driver in the list.

Many device drivers and driver modules can be stored in a single file. Device drivers are created with the Apple III Pascal Assembler, part of the Apple III Pascal System. If you wish to know more about creating device drivers, refer to the *Apple III SOS Reference Manual, the Apple III Driver Writer's Guide,* and the *Apple III Pascal Program Preparation Tools Manual*.

Once the file has been read, the System Configuration Program will ask you for another filename. If you do not want to read more device drivers, press ESCAPE to return to the Configuration menu.

Deleting a Device Driver

The option DELETE A DRIVER allows you to remove one or more device drivers from the current configuration. Since device drivers are always present in the SOS.DRIVER file, and are resident in your system's memory from the time you boot, unused device drivers may be taking up diskette and memory space that could be put to better use storing a program or other data. For example, if you have a text-processing program that does not use graphics, you can remove the .GRAFIX driver from your SOS.DRIVER file in order to get more disk and memory space.

If you choose this option when you have not read any device drivers with the SCP, the screen will show the message

No device drivers have been read.

and the program will return to the Configuration menu. If, however, there are device drivers that you can delete, the screen shows the following:

Photo 3. Delete a Driver

The top section of the screen contains a list of the names of all device drivers currently in memory. The program now waits for you to specify the driver you wish to delete. You can type the number that appears to the left of the driver's name or use the up-arrow and down-arrow keys to move the cursor to the name of the driver and press RETURN. If the driver you wish to delete is part of a driver module, you cannot delete it alone; you must remove the entire module. The program will list the other drivers that will be deleted when you remove the one you indicated and ask if you want to continue. If you want to remove all of the drivers in the module, type the letter Y (for Yes). Otherwise, none of the drivers will be deleted, and you will be asked again which driver to delete.

To keep you from deleting a driver inadvertently, SCP will ask you for confirmation before deleting any driver. If you really want to delete the specified driver, type Y (for Yes); if you have specified the wrong driver by mistake, just type N (for No).

When the specified driver has been removed, the names of the remaining device drivers will reappear at the top of the screen.

Once the driver has been deleted, the System Configuration Program will ask you to select another driver. If you do not want to delete another driver, press ESCAPE to return to the Configuration menu.

The memory used by the System Configuration Program to store a device driver is not recovered when you delete that driver. Each time you add drivers and then remove them, the amount of available memory will diminish. If you find yourself unable to add a new device driver because of the memory lost to deleted drivers, first save the drivers you have on a diskette, then select the QUIT option on the Configuration menu. Next, select the SYSTEM CONFIGURATION PROGRAM option from the Utilities menu to restart the SCP, and reread the file you saved. Now there will be enough memory to add the desired driver.

Editing Driver Parameters

Each device driver has some vital pieces of data, called *driver parameters*, that affect and control its operation. The option EDIT DRIVER PARAMETERS allows you to examine and change some of these parameters.

If you choose this option when you have not read in any device drivers, the screen will show

No device drivers have been read

and the program will return to the Configuration menu. If, however, there are device drivers that you can edit, the screen will show the list of the names of all device drivers currently in memory.

Now you should specify the driver whose parameters you wish to edit. Type the number displayed at the left of the driver's name or use the up-arrow and down-arrow keys to move the cursor to the name of the driver and press RETURN. The System Configuration Program will show you the parameter display for that driver:

Photo 4. Typical Parameter Display

Each driver has several parameters. Some of them can be changed; the others are shown only for your information. The items you can change are displayed at the top of the screen.

To change one of these parameters, type the number displayed at the left of the parameter name or use the up-arrow and down-arrow keys to move the cursor to the parameter name and press RETURN, then type the new value. After you change the parameter, the program will again ask which item to edit; if you do not wish to change any more parameters, press ESCAPE to return to the Configuration menu.

Device Name

The device name is the name SOS uses to gain access to each device driver. The device name must conform to the rules for names: it can be up to 15 characters in length, composed of numbers, letters, and periods. A device name must begin with a period, and the next character must be a letter.

When you select device name as the parameter to change, the program will ask you to type the new device name. Terminate the name by pressing RETURN. If you enter a name that is not a legal device name, the program will again ask you to type a name. When you have typed the name, the parameter display will change to reflect the new name, and the program will again ask you which item to edit.

You can name a device anything you want, but if a program you are using refers to the device by another name, it will not be able to use it.

Device Type and Subtype

The parameters DEVICE TYPE and DEVICE SUBTYPE specify the nature of the device and the way SOS interacts with it. The particular values for the DEVICE TYPE and DEVICE SUBTYPE parameters are important to you only if you are writing a device driver; you will not usually need to know what these parameters do in order to use the driver.

The values of the DEVICE TYPE and DEVICE SUBTYPE parameters contain exactly two hexadecimal characters. Each character may be a digit from Ø to 9 or a letter from A to F. To change one of these parameters, type the number displayed at the left of the parameter name or use the up-arrow and down-arrow keys to move the cursor to the parameter you wish to change and press RETURN. The program will ask you to specify the new value of that parameter. Type the new value and press RETURN.

If you attempt to type a character that is not a digit or a letter from A to F, that character will be ignored. Any lowercase letters you type will automatically be shifted to uppercase. When you have successfully typed the value, the parameter display will be updated to reflect the new value, and the program will again ask which item to edit.

Driver Status

Each device driver can be active or inactive. When SOS loads the system configuration from the SOS.DRIVER file during a boot operation, it does not load any single driver into memory whose status is inactive. If all of the drivers in a module are inactive, the module is not loaded. Even though an inactive driver remains in the SOS.DRIVER file, and its parameters remain set, it is unusable. All active drivers, however, are loaded into memory and can be used.

The maximum number of active drivers in the system is twenty-eight: sixteen block-device drivers and twelve character-device drivers. The ability to make drivers inactive is useful when you have a driver module that contains many drivers, and you don't want to use one or more of the drivers. All the drivers in a module are always loaded with the module, but you can set them inactive.

You can't delete only those drivers you won't be using from the module, but you can set them inactive.

To change the activity status of a driver, select ACTIVITY STATUS and press RETURN. You will be asked whether you want this driver to be active or inactive. Type A to activate the driver or I to make it inactive. The parameter display will change to reflect the new status, and the program will again ask which item to edit.

Editing the Configuration Block

Many device drivers have a special set of parameters that control the operation of the driver. This set of parameters is called the driver's *configuration block*.

The information in the configuration block usually sets the driver's default operation mode. By changing these parameters, you can tailor

the operation of the driver to the particular device you are using. For example, the configuration block for the .PRINTER device driver sets the communications format and data rate the driver uses to send information to the printer. You can change this information with the System Configuration Program. The documentation for each individual driver describes the meaning of each value in the configuration block and tells you how to set the values to perform various functions.

To edit the configuration block for a driver, select the Configuration Block option. The configuration block is optional; if the driver has no configuration block, that item will not appear in the menu. When you select it, the program will display the configuration block:

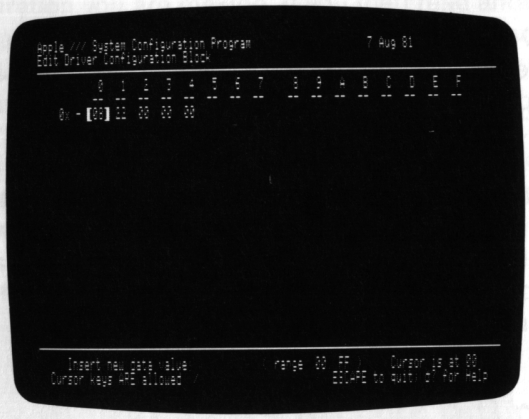


Photo 5. Typical Configuration Block Display

The configuration block contains from 1 to 255 values, arranged in one to 16 rows of up to 16 values each (except the last row). Square brackets inclose the value in the upper left corner of the block; like a cursor, they indicate where the next change will be made. The cursor brackets can be moved to any value in the configuration block by the four arrow keys on the Apple III's keyboard.

Use the arrow keys to move the brackets to a value you wish to change and type the new value. Values consist of two hexadecimal characters, each one a digit from Ø to 9 or a letter from A to F. Type the two characters of the new value and press RETURN. The configuration block will be updated and the brackets will automatically advance to the next value.

You can change this next value by typing a new value as you did before, or you can move the brackets to another value by using the four arrow keys. When you have set all the new values in the configuration block, press ESCAPE to return to the parameter display.

The Comment Field

There is an 8Ø-character comment field stored along with the other device parameters. You can edit the contents of this field in the same way you edit pathnames, by moving the cursor and inserting characters.

You can use this field to help identify driver files you create. For example, you might want to include an explanatory note with an unusual driver file, such as you might create for use with a particular printer. The comment field is for your convenience; you may leave it blank if you prefer.

Changing the System Parameters

Just as the parameters for each individual device driver control the operation of that driver, the system parameters control the operation of the entire operating system.

Four system parameters can be set with the System Configuration Program. These parameters specify:

- the number of Disk III drives you have attached to the Apple III,
- the character set the Apple III will use to display characters on the screen,

- the position and arrangement of the keys on the keyboard, and
- the location of peripheral interface cards in the four peripheral connectors inside the Apple III.

These system parameters are stored with the device drivers in the SOS.DRIVER file on each boot diskette. As with the device drivers, you use the System Configuration Program to change the system parameters.

The first time you read a driver file that contains system parameter information (for example, SOS.DRIVER), that file's system parameters are automatically loaded into the configuration you are making. If you then read another driver file, its parameters will *not* be loaded unless you request it via the ALL option, described below.

To examine or change the system parameters, select the CHANGE SYSTEM PARAMETERS option from the Configuration menu. The screen will show the system parameter display:

Photo 6. Typical System Parameter Display

To change the value of a parameter, first select the parameter you want to change. Then type the new parameter value and press RETURN; if the new value is acceptable, the program will update that parameter in the parameter display and wait for you to select another parameter to be changed. If there are no more changes, you can return to the Configuration menu by pressing RETURN or ESCAPE.

Reading All Parameters

Normally, system parameters are read along with the first driver file that includes system parameters, but you can use the ALL option to read the parameters from any driver file.

To read an entire set of parameters, select the ALL option. The program will ask you to specify the name of the driver file that contains the parameters. If the file you name includes system parameters, the program will read them and they will replace any parameters previously read.

If you try to read system parameters from a file that doesn't have system parameters, the program will display the message

No system parameters associated with file.

and any parameters previously read will remain unchanged.

Number of Disk III Disk Drives

This parameter allows you to set the number of Disk III disk drives you have attached to your Apple III. This number can range from 1 to 4, and includes the disk drive already built into the Apple III.

Peripheral Slot Assignment

Each device driver you have installed with the System Configuration Program can be associated with a peripheral interface card plugged into one of the four peripheral slots inside the Apple III. The Peripheral Slot Assignment option of the system parameter display enables you to tell each device driver the slot number into which you have plugged its peripheral interface card.

SOS allows a peripheral interface card to be used by more than one device driver (one at a time), but a single device driver can not be assigned to more than one peripheral card.

When you select the PERIPHERAL SLOT ASSIGNMENTS option on the system parameter display, you will see a list of all presently installed device drivers and their current peripheral slot assignments:

Photo 7. Peripheral Slot Assignment Display

To the right of the name of each driver is the slot number of the device with which that driver is currently associated. Some drivers will have "n/a" in the place of their slot number; such drivers use only the Apple III's built-in peripherals and are not associated with a peripheral slot.

A driver with an asterisk following its name is inactive; although the activity status is shown in this display, you can change it only with the EDIT DRIVER PARAMETERS option from the Configuration menu.

To change the slot number assigned to a driver, select the driver you want to change. The program will ask you to type the number of the slot you want to assign that driver to; type a number from 1 to 4 and press RETURN. The program will update the display to reflect the change and again ask if you wish to change a slot assignment. To return to the system parameter display, press ESCAPE.

Standard Character Set

The console and graphics drivers have the capacity to display or print text characters. The definition of the shape and appearance of these characters is called the *system character set*. The system character set is part of the system configuration stored in the SOS.DRIVER file. You can change the definition of the standard character set by using the Standard Character Set option of the System Parameters display.

The Standard Character Set option allows you to load a different character set into the current system configuration. Character sets are supplied in diskette files: a few sets are supplied on the Business BASIC and Demonstration diskettes, while others are available as separate products. The format of a character set file is described in Appendix G.

To load a character set, select the STANDARD CHARACTER SET option on the system parameter display. When the program asks for the name of the file that holds the character set, type the complete pathname of the file and press RETURN.

If the file you specified exists and is in the proper format, the System Configuration Program will load it and return to the system parameter display. If the file cannot be found (perhaps you misspelled the name) or is not a valid character set file, the program will tell you so and again ask for a file name. Like the other system parameters, the standard character set is automatically loaded the first time you read the SOS.DRIVER file (or any other file created by the System Configuration Program that includes the character set data) into the current configuration.

When you boot the system, the standard character set is loaded from the file SOS.DRIVER along with the rest of the system configuration. This normally determines the standard character set during all subsequent operation of the system, but it is possible to replace all or part of the standard character set in memory without re-booting the system. Refer to the section on Advanced Techniques in the chapter THE CONSOLE DRIVER.

Keyboard Layout

The Apple III's built-in keyboard generates all 128 ASCII codes in a byte with the high bit set or cleared. Each standard key can generate four different codes: one code if the key is pressed alone, another if the key is pressed in conjunction with the SHIFT key, another if the key is pressed with the CONTROL key, and a fourth code if the key is pressed in conjunction with both the SHIFT and CONTROL keys. The .CONSOLE device driver uses a keyboard layout table to determine the codes that are assigned to each key in its various combinations with SHIFT and CONTROL.

By changing the keyboard layout table, you can change the arrangement of the keys on the keyboard to suit your needs. For example, you could change your keyboard to the Dvorak American Simplified Keyboard layout or to the standard for another language, such as French. Your Apple dealer will have information on how to obtain different keyboard layouts for your Apple III.

The keyboard layout table, like the standard character set, is loaded into the system configuration from a file. To load a keyboard layout, select the option on the system parameter display. The program will ask for the name of the file that holds the keyboard layout; type the complete pathname of the file and press RETURN.

If the file you specified exists and is in the proper format, the System Configuration Program will load it and return you to the system parameter display. If the file does not exist or is not a valid keyboard layout file, the program will tell you and again ask for a name. Like the other system parameters, the keyboard layout is automatically loaded when you read the SOS.DRIVER file (or any other file created by the System Configuration Program that includes the keyboard layout information) into the current configuration.

When you boot the system, the keyboard layout table is loaded from the file SOS.DRIVER along with the rest of the system configuration. This normally determines the keyboard layout during all subsequent operation of the system, but it is possible to replace the keyboard layout table in memory without re-booting the system. Refer to the section on Advanced Techniques in the chapter THE CONSOLE DRIVER.

Generating a New System

Once you have read in all the device drivers you need, deleted the ones you don't need, and set all the driver and system parameters to their proper values, you should save your new system configuration onto a boot diskette. The GENERATE NEW SYSTEM option of the Configuration menu enables you to do this. System generation has two parts: validation of the configuration and the actual writing of the configuration onto a diskette.

When you select this option, the System Configuration Program first examines the configuration you have created to make sure that it is consistent. When this validity check is complete, the program asks you to type the name of the file where you want to save the new system configuration. You may write it to a file with any name you wish. If a file with name you select already exists, the program will give you the option of unlocking and removing the previous file before writing the new one. You will need to do this if there isn't enough room on the boot diskette for both the old file and the new one.

In order for the system configuration to take effect, it must be stored in a file named SOS.DRIVER on a boot diskette. But if you save the configuration you've just created as the file SOS.DRIVER on one of your important boot diskettes, it will destroy the configuration that was formerly on that diskette. If the new configuration is not usable, YOU WILL NOT BE ABLE TO BOOT WITH THAT DISKETTE until you create a proper configuration and store it on that diskette.

Normally, you will have a backup copy, so reconfiguration is not an insurmountable problem. However, if you can't make a backup copy of the boot diskette you're re-configuring, there are two other ways of protecting the old configuration:

1. Copy the SOS.DRIVER file under another name.
2. Use the Utilities Filer to change the SOS.DRIVER file to some other name.

If you need to restore the old configuration, just change the name of the saved file back to SOS.DRIVER. As a general rule, you should make sure you never destroy your original SOS.DRIVER files.

You may want to keep a diskette with a collection of system configurations, under different names and with short descriptions of what they do. Then rather than creating entirely new configurations from scratch each time you change your system, you can load one of the prepared configurations and modify it to get the desired new configuration.

Quitting the Program

When you select the QUIT option from the Configuration menu, this session of the System Configuration Program ends and the program returns to the Utilities menu.

If you try to quit the System Configuration Program without saving your configuration in a diskette file with the Generate New System option, the Apple III will beep and display the message

—GENERATE not performed. Quit (Y/N)?

If you really want to quit without saving the current configuration, type Y (for "Yes") to leave the System Configuration Program and return to the Utilities menu. If you have forgotten to save your configuration, type N (for "No") and the program will display the Configuration menu. You can then save your configuration before quitting.

Using Multiple Configurations of One Driver

The arrangement of the standard device drivers included with the Apple III is fairly straightforward: one device driver controls one input/output device. But occasionally you will have the opportunity to use more than one driver to control a single input/output device, such as when you can have different peripheral devices that you can connect to the same interface device. For example, if you have both a low-speed letter-quality printer and a medium-speed line printer, both of which you can connect to the Apple III's built-in serial interface port, you might need two different versions of the .PRINTER device driver: one to control the letter-quality printer, and one to control the line printer.

To accomplish this, use the System Configuration Program to add the .PRINTER driver to the current configuration, configure the driver for the letter-quality printer, and change its name to .SLOWPRINT (or any other appropriate name). Then load the .PRINTER driver again, configure it for the medium-speed line printer, and change its name to .FASTPRINT . Now you have two copies of the .PRINTER driver, one configured for the letter-quality printer and named .SLOWPRINT, and one configured for the line printer and named .FASTPRINT .

In actual use, you would prepare a document you wish to print, connect the line printer to the Apple III's serial interface port, and print quick draft copies of the document using the .FASTPRINT driver, by specifying .FASTPRINT in the program you are using. Once the document is ready for final printing, you would turn off the Apple III, disconnect the line printer and connect the letter-quality printer to the same serial interface port, then re-boot and print the document using the .SLOWPRINT device driver, without having to reconfigure the system.

3

The Console Driver

The Apple III's built-in keyboard and video display are controlled by the .CONSOLE device driver. Your programs communicate with the keyboard and text screen by means of this driver. When your program opens and reads from .CONSOLE, it reads the data generated by keystrokes on the keyboard; when your program writes to the console, the data it sends is displayed, in text characters, on the Apple III's video screen.

Most language systems open the console driver for reading and writing immediately after the system is booted, and the console driver remains open all the time the system is running. The standard text input and output procedures in each language (such as PRINT, INPUT, and GET in Apple Business BASIC, and READ, READLN, WRITE, and WRITELN in Pascal) automatically perform their operations through the console driver, so you don't usually need to open and operate on it yourself.

In addition to handling normal input and output, the console driver controls the type-ahead and interrupt features of the keyboard, and the cursor motion and text modes of the display. All of the functions of the console are described in this chapter.

What the Console Does

The console driver has two major functions: processing text output and displaying it on the Apple III's video screen, and processing keyboard input and passing it back to programs and language systems.

When the system is booted, the console driver is loaded and initialized. When a language system such as BASIC or Pascal is loaded, it opens the console for input and output. Opening the console for the first time resets it to its default status. Language systems generally perform all their standard input and output through the console.

Screen Output

The Apple III can display a text image on its video display in three modes:

- 24 lines of 40 characters per line, black-and-white characters only;
- 24 lines of 40 characters per line, colored characters on colored backgrounds, 16 colors available;
- 24 lines of 80 characters per line, black-and-white characters only.

Each character displayed on the text screen is composed in a matrix of 56 dots, eight dots high and seven dots wide. Ordinarily, the character set (the design of all the individual characters) is defined at the time the system is booted and can be changed with the System Configuration Program. However, applications programs can replace the character set with a new one by making the appropriate SOS control call to the console driver, as described later in this chapter. Apple Business Basic programs do this via an invokable module, as described in the file DEVICE.DOC on the Business BASIC diskette.

Apple Pascal programs use the procedure UNITSTATUS, described in Appendix H.

When the screen is in one of the black-and-white modes, each position on the screen can display either a character formed by white dots against a black backround, or a character formed by black dots against a white background. When the screen is in the color text mode, each position on the screen can display a character formed by dots in one of 16 colors against a background of any other of the 16 colors. A black-and-white video monitor will display the 16 colors in the color text mode as a progressive grey scale.

The function of the screen output section of the console is to control the placement and appearance of characters on the screen in each mode. The data the console driver uses to do this consists of:

- The *viewport*. The viewport is the rectangular area on the screen where the console places text characters. The console can place new text only in the viewport, so you can change the viewport's size and location to protect data you've already placed on the screen.
- The *cursor position*. This is the position in the viewport where the console will place the next character output to the display. The cursor can be moved up, down, left, or right; it can be moved immediately to a specified position in the viewport; and it can be made visible or invisible.
- The *colors* in which text characters will be displayed on the screen. In the black and white text modes, you can define whether characters are to be displayed as white dots on a black background or as black dots on a white background. In the color text mode you can define the color of the dots that form the character and the color of the background.

You change this data by sending special sequences of characters to the screen. Details on the control of the screen output are given in the following sections of this manual.

Keyboard Input

Whenever you press a key on the keyboard, the console's keyboard handler is automatically invoked to process that keystroke. The keyboard handler translates the keystroke into the corresponding character code, and stores that code in the type-ahead buffer. The type-ahead buffer can normally hold up to 128 character codes.

The keyboard handler performs the translation of keystrokes into character codes by means of the keyboard layout table. This table contains four numeric codes for each standard key on the keyboard: one code for the key pressed alone; another code for the key pressed with CONTROL held down; a third code for the key pressed with SHIFT held down; and a fourth code for the key pressed with both CONTROL and SHIFT held down. Some of these codes may be the same; for example, the standard layout table makes no distinction between 1 and CONTROL-1.

The layout table is loaded into memory from the SOS.DRIVER file when the system is booted; the System Configuration Program allows you to use different keyboard layout tables for different keyboard arrangements. The keyboard layout table cannot be modified once the system is operating.

When a program requests input from the keyboard, the keyboard handler first examines the type-ahead buffer. If there are character codes already in the buffer, they are sent to the program that requested input. If the program needs more character codes than this (either in the number of characters or the lack of a terminating character, such as RETURN), the keyboard handler continues to process keystrokes and send character codes until the proper number of characters (or the terminating character) has been sent.

Each time the keyboard handler sends a character code to the program requesting input, it can also send that character code to the output section of the console: this procedure is called *echoing*. Echoing is usually done automatically, but it can be disabled by a program. Some language systems, such as Pascal, direct the keyboard handler not to echo any characters, so that the language system can control which characters appear on the screen and which ones do not. When echoing is performed by the console, only printable characters are echoed.

Normal Screen Output

When the console driver receives a text string to output, it looks at each character code in the string and decides which of three different ways to handle it.

If the character code is an ASCII control character (with a value of 31 or less), the driver performs a special control function. These control characters and their functions are discussed in the section on Screen Control Codes.

If the code is a valid ASCII text character code (that is, with a value between 32 and 127), the driver places the character in the viewport at the current cursor position.

If the character code has its high bit turned on (making its value between 128 and 255), the driver displays it at the current cursor position, even if it is a control character. Using the standard character set, the driver displays a control character with its high bit on as a two-letter abbreviation of the name of the control character. The driver displays a printing character with its high bit turned on as a normal text character.

After the driver places a character on the screen at the current cursor position, it normally moves the cursor one space to the right. If this would put the cursor beyond the right edge of the viewport, the driver puts the cursor at the beginning of the next line. If the cursor advances beyond the bottom of the viewport, the contents of the viewport are scrolled up one line, and the cursor is placed at the beginning of the new blank bottom line. These movements of the cursor can be controlled individually with a special screen control code; see CURSOR MOTION CONTROL in the section on Screen Control Codes for details.

Text Modes and Character Sets

The size and format of the text screen are called the *text mode*, and the shape and style of the characters on the screen are called the *character set*. Both the text mode and the character set affect all

characters on the screen simultaneously; characters may be displayed in only one character set and in one text mode at any given time.

There are three text modes:

- Mode Ø: 24 lines of 4Ø characters per line, black-and-white characters only (total 96Ø characters);
- Mode 1: 24 lines of 4Ø characters per line, colored characters on colored backgrounds, 16 colors available (total 96Ø characters);
- Mode 2: 24 lines of 8Ø characters per line, black- and-white characters only (total 192Ø characters);

The default text mode is mode 2: an eighty-character wide black-and-white screen. Programs can change this setting by sending the proper screen control code to the console driver.

The Viewport

The viewport is the rectangular area on the text screen that is currently being used by the console driver. The viewport can be as small as one character wide and high, or as large as the entire screen.

The cursor can never move outside the viewport, and the console can never change the contents of the screen outside the viewport. By changing the size and position of the viewport so that you print in only a small area, you can protect anything you've displayed on the rest of the screen.

The viewport acts the same way in all three video modes. The normal size of the viewport is the size of the entire screen; it can be set to any size smaller than the current setting or reset to the size of the screen. The viewport can never be larger than the dimensions of the screen in the current text mode.

Each character position in the viewport can be referred to with a pair of integers. The columns are numbered from left to right, with the leftmost column numbered Ø; the rows are numbered from top to bottom, with the top row in the viewport numbered Ø. The upper-left corner of the viewport is always numbered Ø,Ø regardless of its position or the size of the viewport; all other positions in the viewport are numbered relative to the upper-left corner. The character positions are the same as absolute screen positions only if the upper-left corner of the viewport is set to the upper-left corner of the screen.

Cursor Motion and Controls

The cursor normally moves in the same way that English text is written. As each printing character is placed on the screen, the cursor advances to the right. If the cursor is moved past the right edge of the viewport, it automatically moves to the left edge of the viewport and moves down one line; this is called *wraparound*. If the cursor is moved past the bottom of the viewport, the contents of the viewport are scrolled up one line and the cursor moves to the beginning of the newly blank bottom line.

When moving backwards, the cursor motion is entirely analogous to its forward motion. If the cursor is moved past the beginning of a line (the left edge of the viewport), it is placed at the right edge of the viewport and moved up one line. If the cursor is moved past the top of the viewport, the contents of the viewport are scrolled down one line and the cursor is placed at the end of the newly blank top line.

There are three ways to cause the cursor to move. The first is to send a printing character to the console: the cursor will advance automatically after the character is printed. The second way is to send one of several screen control codes to the console to move the cursor left, right, up, down, to the beginning of the next line, or to the upper-left corner of the viewport. The third way to move the cursor is to send other screen control codes followed by numerical arguments that specify the new cursor position. A description of all screen control codes is given in the next section.

Most programming languages on the Apple III also provide a more convenient way to move the cursor. For example, you can use the HPOS and VPOS reserved variables in Apple Business BASIC to set and read the position of the cursor directly; other languages have similar facilities. All of them operate by sending the proper screen control codes to the console.

The automatic motions of the cursor can be enabled or disabled. You can specify that the cursor remain fixed after printing a character on the screen and not move one space right; you can specify that the cursor stop at the right or left border of the viewport, and not wrap around; you can tell the console to move the cursor down one line automatically when it performs a carriage return, or you can tell it just to move the cursor to the beginning of the same line it is on; and you can prevent the viewport from scrolling if the cursor is moved beyond the bottom or top edge. All of these cursor motions can be controlled with a screen control code (see the CURSOR MOTION CONTROLS code in the following section).

Screen Control Codes

The ASCII control characters with values between Ø and 31 are recognized by the console as *screen control codes*. When the console receives one of these codes during output, it does not place a character on the screen; instead, it performs a control function that changes the appearance or behavior of the viewport or cursor.

Some of the screen control codes require one or more *arguments*. These are extra characters immediately following the control code that do not perform their regular functions, but instead supply extra information to a function invoked by the code. For example, the FOREGROUND COLOR screen control code (code 19) requires one argument whose value represents the color that is to be used to display characters. A segment of a program to set the foreground color to dark blue (color 2) might be written in BASIC as

```
30 SETFORE=19 : DARKBLUE=2
40 PRINT CHR$(SETFORE);CHR$(DARKBLUE);
```

and in Pascal might read

```
var:
  g_array:array[0..20] of 0..255;
begin
  g_array[0]:=19; g_array[1]:=2;
  unitwrite(1,g_array,2,,12);
end;
```

(Note the use of the UNITWRITE procedure in the Pascal example. You cannot use WRITE or WRITELN here, because the values being transmitted could be one of the control-character values that Pascal traps, that is, 13 or 16. You should limit the use of UNITWRITE to small procesures that can be easily replaced if you wish to run the program on another Pascal system.)

There are seven screen control codes that require one argument character and one control code (ABSOLUTE CURSOR POSITION) that requires two argument characters.

The following is a list of the 32 screen control codes and their functions. Each code number is followed by the ASCII name and abbreviation for that code.

Code 00, "NUL" (Null)
This character has no effect on screen output.

RESET VIEWPORT
Code 01, "SOH" (Start Of Header)
Resets the viewport to the size of the current text screen. Leaves the cursor in the same position on the screen; its coordinates relative to the origin of the viewport may change. This function also saves the previous viewport setting, cursor position, screen mode (see code 16), cursor motion controls (see code 21), normal/inverse mode (see codes 17 and 18), and text colors (see codes 19 and 20) so they can be restored by the RESTORE VIEWPORT command (see below).

VIEWPORT TOP
Code Ø2, "STX" (Start of Text)

Sets the upper-left corner of the viewport to the current cursor position. This position will now have coordinates (Ø,Ø), and all cursor movements will be made relative to those coordinates.

VIEWPORT BOTTOM
Code Ø3, "ETX" (End of Text)

Sets the lower-right corner of the viewport to the current cursor position.

RESTORE VIEWPORT
Code Ø4, "EOT" (End of Transmission)

Restores the viewport and cursor position to the state they were in when the last RESET VIEWPORT command was issued (above). Also restores the screen mode, cursor motion controls, normal/inverse mode, and text colors to their values at the time of the last RESET VIEWPORT command. If there has been no RESET VIEWPORT command since the console was first opened, these parameters will be set to their default values.

CURSOR ON
Code Ø5, "ENQ" (Enquiry)

Makes the cursor visible. The cursor is a character-size square that is normally the inverse of the character it is positioned on. The cursor is usually not displayed during output; the console turns the cursor on whenever input is requested, then restores it to its former state (determined by the program) when the input request is complete.

CURSOR OFF
Code Ø6, "ACK" (Acknowledge)

Makes the cursor invisible (see above). All output is still processed normally.

SOUND BELL
Code Ø7, "BEL" (Bell)

Sounds a short beep on the Apple III's built-in speaker. If a miniature phone plug is inserted into the AUDIO OUT jack on the back of the Apple III, no sound will be generated by the speaker, but any device attached to that plug will receive the audio signal of the beep. The driver resumes normal processing immediately after the beep is started: it does not wait until the beep is finished. If you output another BEL code while a beep is in progress, the second code will be ignored.

MOVE CURSOR LEFT
Code Ø8, "BS" (Backspace)

Moves the cursor position one space to the left. Wrapping around and scrolling are performed in accordance with the setting of the cursor motion controls (see below).

MOVE CURSOR RIGHT
Code Ø9, "HT" (Horizontal Tabulation)

Moves the cursor position one space to the right. Wrapping and scrolling are performed in accordance with the setting of the cursor motion controls (see below).

MOVE CURSOR DOWN
Code 1Ø, "LF" (Line Feed)

Moves the cursor down one line. Scrolling is performed in accordance with the setting of the cursor motion controls (see below).

MOVE CURSOR UP
Code 11, "VT" (Vertical Tabulation)

Moves the cursor up one line. Scrolling is performed in accordance with the setting of the cursor motion controls (see below).

HOME CURSOR
Code 12, "FF" (Form Feed)

Moves cursor to the upper-left corner of the viewport. Does not clear any portion of the screen or change viewport setting.

RETURN CURSOR
Code 13, "CR" (Carriage Return)

Moves cursor to the beginning of the current line (the left edge of the viewport). A line feed may be issued automatically after the return, in accordance with the setting of the cursor motion controls (see below). If a line feed is issued, scrolling occurs also in accordance with the cursor motion controls.

TURN SCREEN OFF
Code 14, "SO" (Shift Out)

Disables the Apple III's video generator. The video monitor shows a blank screen regardless of the contents of any text or graphics screen, although the information on the text screen is not lost, and new data can still be written. The image will be restored to the screen when a TURN SCREEN ON command is issued (see below). This command has no effect if the screen is already off.

TURN SCREEN ON
Code 15, "SI" (Shift In)

Enables the Apple III's video generator. The video monitor will display text in the currently selected text mode. If one of the graphics modes was in effect, sending the TURN SCREEN ON command to the console will return the display to the text mode. This command has no effect if the text screen is already being displayed.

TEXT MODE One argument
Code 16, "DLE" (Data Link Escape)

Sets the text mode in which the console is to place and display text information. The next character following the control code specifies the text mode to set. These are the text modes and their specification characters:

Mode	Character
40x24 Black-and-white	
40x24 Color	
80x24 Black-and-white	

A specification character whose least significant bits have the value 3 will select mode 2. Only the two least significant bits of the character are used; the upper six bits are reserved in order to assure compatability with future versions of .CONSOLE and should be set to zero.

The TEXT MODE command takes effect immediately after the console receives the specification character and affects the entire screen. Any text sent to the screen after a TEXT MODE command will always be processed in the new text mode. If, however, the Apple III's screen is in a graphics mode, the new text mode will not be visible until a TURN SCREEN ON command is sent to the console.

The TEXT MODE command adjusts the setting of the viewport only when the viewport will not fit on the new text screen. When changing from the 80-column mode to one of the 40-column modes, the left and right borders of the viewport will be adjusted so that they fit into 40 columns. When changing from one of the 40-column modes to the 80-column mode, the viewport remains 40 columns wide. You should make the appropriate changes to the viewport setting whenever you change text modes.

This command changes the way the Apple III interprets the data that is being displayed when the command is given. Changing text modes with text already on the screen will not modify the data producing that text to make it appear the same in the new mode. For example, changing from colored text mode to 80-column mode will fill every other column on the screen with seemingly random characters; these characters were the color information in the colored text mode. Similarly, changing from 80-column mode to colored text mode will cause every other column of characters on the screen to vanish and affect the color settings of the remaining columns; therefore, it is advisable to clear the screen after selecting a new text mode.

NORMAL

Code 17, "DC1" (Device Control 1)

Specifies that all subsequent characters will be displayed as white characters on a black background (or, in color mode, as characters of the foreground color on a field of the background color). Does not affect any characters already on the screen.

INVERSE
Code 18, "DC2" (Device Control 2)

Specifies that all subsequent characters will be displayed as black characters on a white background (or, in color mode, as characters of the background color on a field of the foreground color). Does not affect any characters already on the screen.

FOREGROUND COLOR One argument
Code 19, "DC3" (Device Control 3)

Sets the foreground color (used only in the color text mode) to the value specified by the next following character. All subsequent characters will be displayed in the new color; characters already on the screen are not affected. Only the lower four bits of the argument character are recognized. The upper four bits are reserved: in order to assure compatability with future versions of .CONSOLE, they should be set to zero.

Here are the sixteen colors and their color values:

Color	Value	Color	Value
Black	0	Brown	8
Magenta	1	Orange	9
Dk. Blue	2	Grey 2	10
Purple	3	Pink	11
Dk. Green	4	Green	12
Grey 1	5	Yellow	13
Med. Blue	6	Aqua	14
Light Blue	7	White	15

BACKGROUND COLOR One argument
Code 2Ø, "DC4" (Device Control 4)

Sets the background color (used only in the color text mode) to the value specified by the next following character. Characters and their associated colors are given in the table above. All subsequent characters will be displayed in the new color; characters already on the screen are not affected. Only the lower four bits of the argument character are recognized. The upper four bits are reserved: in order to assure compatability with future versions of .CONSOLE, they should be set to zero.

CURSOR MOVEMENT CONTROL
Code 21, "NAK" (Negative Acknowledge)

Sets the controls that determine various movements of the cursor according to the value of the immediately following character. Only the lower four bits of the character are significant. The upper four bits are reserved: in order to assure compatability with future versions of .CONSOLE, they should be set to zero.

The four controls are Advance, Line Feed, Wrap-around, and Scroll. When Advance is active, the cursor moves one space to the right after each character is placed on the screen; when inactive, the cursor remains at the same position. When Line Feed is active, the cursor performs a line feed after every return; when inactive, no automatic line feed is performed. When Wrap is active, an attempt to move the cursor beyond the right or left edge of the viewport causes the cursor to be placed at the opposite edge of the next or previous line, respectively; when inactive, the cursor remains at the edge of

the viewport. When Scroll is active, an attempt to move the cursor beyond the top or bottom line of the viewport causes the contents of the viewport to be scrolled down or up, respectively, and the cursor to be placed on the new top or bottom line; when inactive, the cursor remains at the top or bottom of the viewport.

The setting of the four controls is determined by the lower four bits of the specification character:

Control	Bit	Default Value
Advance	Ø	1
Line Feed	1	Ø
Wrap	2	1
Scroll	3	1

If a given bit has a value of 1, the control associated with it is enabled; if its value is Ø, the control is disabled. If these control bits have not been set since the system was booted, they will have the default values shown. A table in Appendix A gives the 16 distinct argument characters and their effects. The upper four bits of the specification character are reserved: in order to assure compatability with future versions of .CONSOLE, they should be set to zero.

SCREEN SYNCHRONIZATION

Code 22, "SYN" (Synchronous idle)

Causes the console to delay further processing until the video generator has finished displaying one complete frame on the video display. The video generator produces 6Ø frames each second; a SYN code can thus cause a delay of up to 1/6Ø of a second.

One application of this control code is in programs that erase and redraw portions of the screen in a fixed cycle, to perform animation or other special effects. Sometimes the difference between the timing of the program's output and the timing of

the video generator produces a distracting flicker of the screen. Sending a SYN code to the console as part of the erase/redraw cycle will synchronize the program with the video display and reduce or eliminate the flickering.

Another use of the SYN code is in timing loops. Since the video generator always produces 60 frames per second regardless of the screen setting, sending a string of 60 SYN codes will cause a delay of one second. You can use this to build fairly accurate timing loops into user programs.

HORIZONTAL SHIFT One argument
Code 23, "ETB" (End Transmission)

Shifts the text inside the viewport according to the value of the immediately following argument character. The argument character is interpreted as an eight-bit two's complement value; if it is positive, the contents of the viewport are shifted right the number of columns equal to the value of the character. If the argument is negative (binary value greater than or equal to 128), the contents of the viewport are shifted left the number of columns equal to the negative value of the character.

The shifted characters are moved directly to their destination location, not scrolled one column at a time. The space vacated by the shifted characters is set to blanks; characters shifted out of the viewport are removed from the screen and are not recoverable. If the shift distance is greater than or equal to the width of the viewport, the shift command has the effect of clearing the viewport.

HORIZONTAL POSITION One argument
Code 24, "CAN" (Cancel)

Moves the cursor horizontally to the column specified by the argument character. The character following the control code is used to determine the new cursor position in the viewport. If the value of the character is greater than the width of the viewport, the cursor is placed at the right edge of the viewport.

VERTICAL POSITION One argument
Code 25, "EM" (End of Medium)

Moves the cursor vertically to the specified row. The following character is used to determine the new cursor position in the viewport. If the value of the character is greater than the height of the viewport, the cursor is placed on the bottom line of the viewport.

ABSOLUTE POSITION Two arguments
Code 26, "SUB" (Substitute)

Moves the cursor to the specified row and column. The first following character is used to determine the new horizontal position; the second following character is used to determine the new vertical position. If the value of the first character is greater than the width of the viewport, the cursor is placed at the right edge of the viewport; if the value of the second character is greater than the height of the viewport, the cursor is placed on the bottom line of the viewport.

Escape
Code 27, "ESC" (Escape)

This character currently has no effect on screen output. It is reserved for future enhancement of the system.

CLEAR VIEWPORT
Code 28, "FS" (Field Separator)

Moves the cursor to the upper-left corner of the viewport and sets the contents of the viewport to space characters (ASCII code 32).

CLEAR TO END OF VIEWPORT
Code 29, "GS" (Group Separator)

Clears the contents of the viewport from the cursor to the end of the viewport: that is, from the cursor position to the end of the line and all lines below the cursor. The cursor is not moved.

CLEAR LINE
Code 3Ø, "RS" (Record Separator)

Moves the cursor to the left edge of the viewport and clears the entire line the cursor is on.

CLEAR TO END OF LINE
Code 31, "US" (Unit Separator)

Clears the contents of the line the cursor is on from the cursor position to the right edge of the viewport. The cursor is not moved.

Since the space character usually appears as a blank space of the background color, the screen area cleared by each of the CLEAR commands, the SHIFT command, and the SCROLL portion of CURSOR MOVEMENT CONTROL is normally set to black (or the background color in color mode). If inverse mode is in effect (see DC2, above), then all of these commands will set the cleared screen to white (or the foreground color).

Keyboard Input

When a program or language system requests input from the console, several things happen. First, the cursor is made visible at the current cursor position; it appears as the inverse of the character at that position. If the character is a space (a common condition), the cursor appears as a solid white block (in the color text mode, as a block of the foreground color).

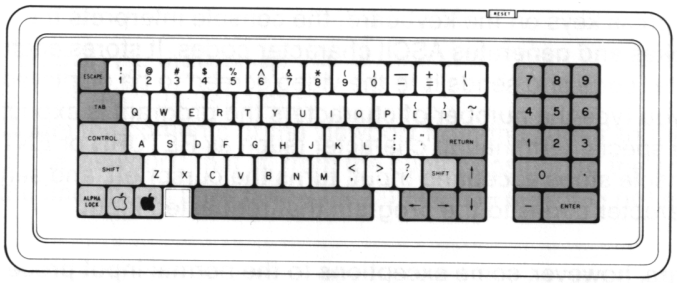
As you press keys on the keyboard, the console interprets the keystrokes and generates ASCII character codes. It stores each character code and sends it to the video screen to be displayed. When you type the number of characters the program is expecting, or press a special terminating character (such as RETURN or ENTER), the console stops accepting input, turns the cursor off, and sends all the character codes to the program that requested input.

There are, however, some exceptions to the normal input process. *Type-ahead* enables you to type characters on the keyboard before the program is ready to process them; the *editing keys* and *escape mode* let you correct typographical errors before the characters are sent to the program; and *console control keystrokes* enable you to control many aspects of the operation of the console. All of these are described in the following sections.

The Keyboard

There are 73 keys on the Apple III's keyboard. They are logically divided into three groups: standard keys, special keys, and modifier keys.

The standard keys are the 26 alphabetic keys, the 10 numeric keys, and the 11 symbol keys on the main keyboard. Each of these can generate up to four distinct ASCII codes, depending upon which modifier keys are held down when you press the standard key. The keyboard layout table (specified with the System Configuration program) specifies the four codes associated with each standard key: one for the key pressed alone, another for the key pressed with CONTROL held down, a third for the key pressed with SHIFT held down, and a fourth code for the key pressed with both SHIFT and CONTROL held down.



Standard keys: Special keys: Modifier keys:

The special keys are the ESCAPE, TAB, RETURN, the spacebar, and the four arrow keys on the main keyboard, and all keys on the numeric keypad. The special keys always generate the same ASCII codes, regardless of the condition of the SHIFT and CONTROL keys. The keyboard layout table does not define codes for these keys.

The modifier keys are the CONTROL key, both SHIFT keys, the ALPHA LOCK key, and both Apple keys. The CONTROL and SHIFT keys alter the codes produced by the standard keys as described above. The ALPHA LOCK key (which locks in the down position) has the same effect as the SHIFT key, but affects only the alphabetic keys.

The Open Apple key, when held down, sets the high bit of the ASCII codes generated by the keyboard. It acts independently of the SHIFT and CONTROL keys, and affects both the standard and the special keys. The Open Apple effectively adds 128 to the numeric code of any keystroke.

If you press the Solid Apple key while another key is down, it acts as a fast-repeat key. Each standard or special key, if held down for more than half a second, will begin to repeat automatically; pressing Solid Apple increases the rate at which the other key repeats. When used this way, the Solid Apple key does not affect the ASCII code generated by a key. If you press the Solid Apple key *before* pressing another key, it only causes a bit to be set in the second byte of keyboard data, and does not act as fast-repeat. Programs can use it as a different kind of shift key.

Tables of the codes generated by each key appear in Appendix A; the bit assignments in the two bytes of keyboard data are given in Appendix G.

Type-ahead

The console's type-ahead abilities allow you to type information into the Apple III before the program you are using is prepared to accept and process that information. This means that you don't have to wait for the computer to finish executing one command before you can issue the next one; you can type at your normal speed, and the computer will catch up as soon as it can.

Each time you press a key while the console is not expecting input, it stores that key's character code in the type-ahead buffer. When a program requests input from the console, the console first moves all characters in the type-ahead buffer, one at a time, into the input buffer. It also echoes them to the display at this time. When the type-ahead buffer has been emptied, the console resumes normal input.

If the characters in the type-ahead buffer satisfy the input request, any characters remaining in the type-ahead buffer will be saved and used for the next input request.

The usual limit to the number of characters you can enter into the type-ahead buffer is 128. If you type 128 characters without having any processed by the computer, each subsequent character you type will be ignored and the console will sound a short beep at each keystroke. As soon as the computer begins processing the characters in the type-ahead buffer, you can enter more characters.

A console control keystroke can be used to erase the contents of the type-ahead buffer, in case you have entered information that you do not wish the Apple III to process. See the section that describes console control keystrokes, later in this chapter, for details.

Some programs you might use on the Apple III reduce the size of the type-ahead buffer or disable type-ahead completely (by using the advanced techniques described at the end of this chapter). When one of these programs is running, any characters you type when the Apple III is not waiting for them will be lost.

Backspace, Retype, and Cancel

At any time before you complete an input request and the console sends the characters you've typed back to the program that requested input, you can edit the characters you've typed to correct any typographical errors you might have made. Three special functions help you edit your input line: backspace, retype, and cancel.

The backspace function, usually activated by the left-arrow key or CONTROL-H keys, deletes a single character from the input buffer. Each time you press the backspace key, the last character in the input buffer is removed, and the cursor is moved back one space. Normally this moves the cursor to the position of the character that was just deleted. If there are no characters in the input buffer, none can be deleted, and the cursor is not moved.

There are two kinds of backspace: destructive and nondestructive. Both kinds remove the last character from the input buffer, but destructive backspace erases the deleted character from the screen as the cursor moves over it, while nondestructive backspace leaves the character on the screen. The kind of backspace in effect depends on the particular program or language system you are using.

The retype function, usually invoked by the right arrow key, puts characters back into the input buffer after you have removed them with a nondestructive backspace. When you press the retype key, the console takes the ASCII code of the character on the screen at the cursor position and enters that code into the buffer. It is just as if you had typed the keystroke that generated that character.

These two functions are used in correcting simple typographical errors. You use the backspace key to move the cursor back to the location of the error, type the correct character, then use the retype key to move the cursor ahead to its original position. The backspacing operation removes the characters it passes over from the input buffer, and the retype operation puts them back, in the original order. A complete description of the procedure is given in the *Apple III Owner's Guide*.

The retype key is also useful in conjunction with escape mode (see below). Note: the retype key does not work on some programs and language systems.

The cancel function, CONTROL-X, removes all characters from the input buffer. Every character that has been entered (either from the keyboard or from the type-ahead buffer) since the beginning of the input request will be deleted. This allows you to cancel an erroneous line and start with a fresh line, without having to backspace all the way to the beginning of the line.

Like backspacing, cancelling can be destructive or nondestructive. Destructive cancel issues one destructive backspace for every character in the input buffer, thus removing all characters in the input line from the screen. Nondestructive cancel simply prints a backslash (\) at the end of the cancelled input line and places the cursor at the beginning of a fresh line. As with the other keyboard features, the kind of cancel (if any) depends on the particular program or language system you are using, and can be changed: see the section on Advanced Techniques.

Cursor Commands: Escape Mode

When the console is accepting input from the keyboard, you can use the cursor commands to move the cursor around on the screen, set or reset the viewport, and clear various portions of the screen. None of the cursor commands you issue are placed in the input buffer, and they are not sent as input to the program you are using.

Cursor commands are useful for data entry and program editing in languages like BASIC. They are not used with Pascal, which disables the escape mode and uses its own screen-editing commands.

To issue cursor commands while the console is expecting input, press the ESCAPE key. If cursor commands are allowed, the cursor will change into a black plus sign on a white background (or a white plus sign on a black background, if the character the cursor is on is inverse):

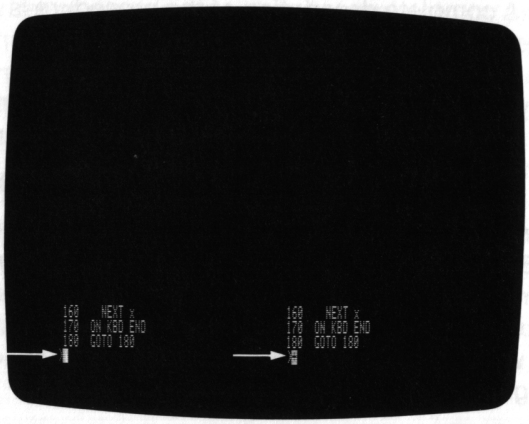
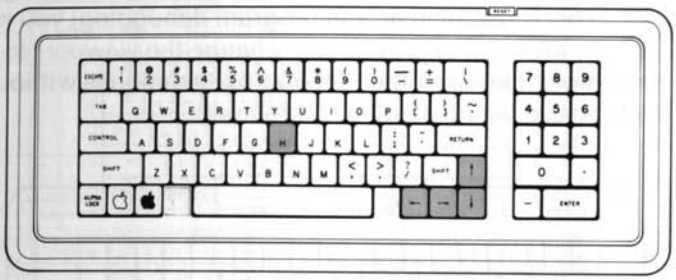


Photo 8. Normal Cursor Cursor Command

The cursor will remain like this until you turn off the cursor command mode.

There are eleven cursor commands. Each one corresponds directly with a screen control code (see the previous section); the cursor command mode is, in fact, just a simple way to issue these eleven screen control codes from the keyboard. To turn off cursor command mode, press any key that is not a cursor command.

In cursor command mode, each of the four arrow keys moves the cursor one space in the direction of the arrow. The left and right arrows do not perform their normal functions as backspace and retype; that is, the input buffer is not modified by cursor commands. The left and right arrow keys regain their backspace and retype functions as soon as you turn off cursor command mode.

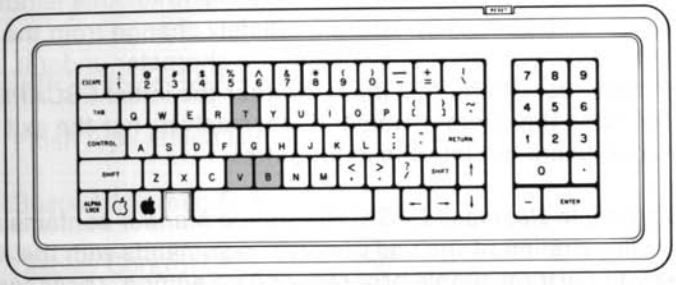


Cursor commands for moving the cursor

Typing the letter H (for Home) in cursor command mode will move the cursor to the upper-left corner of the viewport.

You should be careful when using cursor commands in the middle of an input line, especially when using the backspace or cancel functions. These functions move the cursor to reflect the changes they make in the input line; if you move the cursor away from the input line with cursor commands, then any subsequent backspace or cancel functions will not indicate accurately the characters that they remove from the input buffer.

There are three cursor commands that clear various portions of the viewport. Pressing S (for Screen) in cursor command mode will clear the contents of the viewport and move the cursor to the upper-left corner. Pressing P will clear from the cursor position to the end of the viewport without moving the cursor; pressing L (for Line) will clear from the cursor position to the end of the line the cursor is on, again without moving the cursor. For detailed descriptions of the operation of the clearing commands, see the screen control codes 28, 29, and 31 in the previous section.



Cursor commands to set the viewport

Pressing V (for Viewport) in cursor command mode resets the viewport to the full dimensions of the text screen, without moving the cursor. It also saves the former viewport setting (and other important information) so that it can be restored later by the RESTORE VIEWPORT screen control code. The restoration function, however, is not available in cursor command mode. See the screen control codes Ø1, Ø2, and Ø3 in the previous section for details of the viewport controls.

Here is a summary of the cursor commands, their functions, and their equivalent screen control codes:

Character	Function	Control Code
Left Arrow	Move cursor left	Ø9 HT
Right Arrow	Move cursor right	Ø8 BS
Up Arrow	Move cursor up	11 VT
Down Arrow	Move cursor down	1Ø LF
H or h	Home cursor	12 FF
S or s	Clear viewport	28 FS
P or p	Clear to end of viewport	29 GS
L or l	Clear to end of line	31 US
V or v	Reset viewport	Ø1 SOH
T or t	Set viewport upper-left	Ø2 STX
B or b	Set viewport lower-right	Ø3 ETX

These functions can be performed with console control keys:

- Turn Video Output On and Off,
- Flush Type-ahead Buffer,
- Suspend Screen Output,
- Display Control Characters, and
- Flush Screen Output.

One function, Flush Type-ahead Buffer, is performed each time you press the proper keys. The other functions are *toggled*: typing the control key once turns the function on, and typing it again turns the function off.

Toggle Video Output, CONTROL-5 on the numeric keypad, turns the video output on and off. You can still write data to the console when the video output is off, but it will not be visible until the video is turned on. Note that an input request or an I/O reset from a program automatically turns the video on. The Apple III runs programs about 20 percent faster with the video turned off than it does with video on. Thus, you can manually speed up portions of your programs that do not require video output by using this control.

Flush Type-Ahead Buffer, CONTROL-6 on the numeric keypad, removes all characters in the type-ahead buffer. You use this if you have typed a lot of data into the type-ahead buffer and realize you made an error; pressing CONTROL-6 erases the information before the console can put it into the input buffer and echo it to the screen. If the characters have already been moved from the type-ahead buffer into the input buffer, CONTROL-6 has no effect.

Suspend Screen Output, CONTROL-7 on the numeric keypad, suspends all screen output. When you press CONTROL-7, the console is prevented from placing any new information on the screen. If a program attempts to put text onto the screen, it will be stopped until you enable screen output by pressing CONTROL-7 again. The second time you press CONTROL-7, program output will resume from the point where it was stopped.

Display Control Characters, CONTROL-8 on the numeric keypad, affects the way control characters are output. ASCII control characters sent to the console are usually interpreted as screen control codes (see the previous section) and affect the appearance of the screen, cursor, and viewport. After you press CONTROL-8, any control characters sent to the screen will not be interpreted as screen control codes, but instead will be displayed as characters on the screen. This is sometimes useful in debugging. The appearance of the control characters is defined by the system character set; in the normal character set, control characters are displayed as two-letter abbreviations of the names of the characters. To enable control characters to perform their normal screen control functions, press CONTROL-8 again.

All screen control codes will have no effect after you press CONTROL-8. This includes the screen characters issued by the cursor command mode: in other words, ESCAPE mode isn't useful when CONTROL-8 is in effect.

Flush Output, CONTROL-9, controls screen output, but differently from CONTROL-7, above. When you press CONTROL-9, the console stops processing output, but programs are not halted: any characters sent to the console after you press CONTROL-9 are ignored. When you press CONTROL-9 again, the console resumes processing output. Note that an input request will automatically cancel the effect of CONTROL-9 and cause the console to resume normal output processing.

Here is a summary of the console control keystrokes:

Keystroke	Function
CONTROL-5	Toggle Video Output (toggle)
CONTROL-6	Flush Type-ahead Buffer
CONTROL-7	Suspend Screen Output (toggle)
CONTROL-8	Display Control Characters (toggle)
CONTROL-9	Flush Output (toggle)

Advanced Techniques

Whenever you send a control character to the console driver, the language system you are using actually transmits the character by means of a system call. System calls are the normal method that assembly-language programs (including interpreters and language systems) use for communicating with files and devices on the Apple III.

Instead of sending control characters as part of the output character stream, you can control the operation of the console by issuing your own system calls. The particular system calls that you use to control device drivers are the D_STATUS and D_CONTROL calls.

Interpreters (such as BASIC) and language systems (such as Pascal) provide more convenient ways of communicating with files and devices, so that user programs don't ordinarily need to resort to system calls. However, some language systems on the Apple III provide ways for you to make SOS calls directly to the system. For example, Apple III Pascal includes a procedure called UNITSTATUS that is used for both D_STATUS and D_CONTROL system calls. The use of this procedure is described in Appendix H.

These system calls require three parameters, like this:

D_STATUS (device number, status code, status list)

D_CONTROL (device number, control code, control list)

where device number specifies the device you want to get status information from or send control information to; status code and control code specify the particular information to get or to send; and status list and control list specify the location in the program's memory space where the information is or will be put. For more details on using system calls from an assembly language program, refer to the *Apple III SOS Reference Manual*.

Console Status Requests

The following list gives the status code and the contents of the status list for each console driver status request. Note: status bytes are often given as hexadecimal values, indicated here by a dollar sign ($) prefix, e.g., $8Ø is hexadecimal 8Ø, equal to decimal 128.

Status code: Ø (No Operation)
Status list: (nil)

This status request does nothing.

Status code: 1 (Preserve Status Table)
Status list: Buffer size, Buffer

Copies the status table of the console into the buffer portion of Status list. The status table contains the current state of the console options; once the status is preserved, you can change the console options as much as you like and still be able to restore them by issuing a Restore Status Table control request.

The first byte of the status list is the buffer size. You set Buffer size to the number of bytes available in the buffer before executing the request. For the console driver, the buffer must be at least 9Ø bytes long. The driver will store the status table into the buffer and put the number of bytes actually used into Buffer size. The contents of the status table are for the console's use only: you should not attempt to do anything other than preserve and restore the status table.

Status code: 2 (Termination Character)
Status list: Line status; Line character

Returns the line-termination status and character. If Line status is equal to $8Ø, then pressing the character given in Line character will terminate an input request; if Line status equals $ØØ, only satisfying the proper character count will terminate an input.

Status code: 3 (Keyboard Mode)
Status list: Mode status

Returns the keyboard mode. If Mode status equals $00, then a read request to the keyboard returns a single byte of ASCII code for each keystroke. If Mode status equals $80, a read request returns a byte of ASCII code plus an additional byte of keyboard data. The format of the additional data byte is given in Appendix G.

Status code: 4 (Type-ahead Buffer Size)
Status list: Buffer size

Returns the size of the type-ahead buffer as the number of keystrokes it can store. If type-ahead is disabled, this status request will return a buffer size of zero.

Status code: 5 (Buffered Keystroke Count)
Status list: Keystroke count

Returns the number of keystrokes currently stored as characters in the type-ahead buffer.

Status code: 6 (Attention Event)
Status list: Priority, Event ID, Event-handler address, Attn char

Returns the current Attention Event parameters. The Attention event monitors keystroke data and tests for the character defined as the Attention character. When it detects the Attention character, the console driver flushes the type-ahead buffer and, if type-ahead is enabled, puts the character into the buffer. The console also turns off the event so that it cannot recur until the program reactivates it. Then the console passes control to the Event-handler address, as if it had executed a JSR instruction. If the Any-key event happens at the same

time as the Attention event, the Any-key event takes precedence. Refer to the *Apple III SOS Reference Manual* for a complete description of Events.

The Event-handler address is a three-byte address: low byte, high byte, and bank. The Event-handler routine must be present at this address throughout the period when it is possible for this event to be active. If the event occurs when some other routine has been loaded or relocated to this address, the system will fail: you will have to reboot it and start your program over from the beginning. If your program cannot guarantee that the Event-handler routine will remain in memory, you should not use events.

Status code: 7 (Reserved)
Status list: (nil)

This request code is reserved for future assignment.

Status code: 8 (Any-Key Event)
Status list: Priority, Event ID, Event-handler address

Returns the current Any-Key Event parameters. The Any-Key Event is triggered by the pressing of any key. When the Any-key Event detects a character, the console driver flushes the type-ahead buffer and, if type-ahead is enabled, puts the character into the buffer. The console also turns off the event so that it cannot recur until the program reactivates it. Then the console passes control to the Event-handler address, as if it had executed a JSR instruction. If the Any-key event happens at the same time as the Attention event, the Any-key event takes precedence. Refer to the *Apple III Technical Reference Manual* for a complete description of Events.

The Event-handler routine must be present throughout the period when it is possible for this event to be active. If your program cannot guarantee that the Event-handler routine will remain in memory, you should not use events. See the warning given under status code 6.

Status code: 9 (Reserved)
Status list: (nil)

This request code is reserved for future assignment.

Status code: 10 (No Wait Input)
Status list: No-Wait status

Returns the status of the no-wait mode. If No-Wait status equals $80, then all input requests will return only characters that were in the type-ahead buffer at the time of the input request; the console will not wait until the user types the termination character or satisfies the character count. If No-Wait status equals $00, input is processed normally.

Status code: 11 (Screen Echo)
Status list: Screen-Echo status

Returns the status of the screen-echo mode. If Screen-Echo status equals $00, then no keypresses will be echoed to the screen; in addition, there will be no cursor display, no ESCAPE mode, no character copy, and no cursor movement during backspace and cancel. If Screen-Echo status equals $80, input characters will be echoed to the screen, and the other functions listed will all operate normally. If Screen-Echo status equals $00 or $80, any control characters typed on the keyboard will not be echoed to the screen. If Screen-Echo status equals $C0, then, in addition to normal character echoing, control characters will also be echoed to the screen.

Status code: 12 (Retype)
Status list: Retype status

Returns the status of retype mode. If Retype status equals $00, retype is disabled. If Retype status equals $80, retype is enabled.

Status code: 13 (Backspace)
Status list: Backspace status

Returns the status of the backspace function. If Backspace status equals $00, the backspace function is disabled. If Backspace status equals $80, then backspace is enabled and nondestructive; if Backspace status equals $C0, backspace is enabled and destructive.

Status code: 14 (Cancel)
Status list: Cancel status

Returns the status of the cancel function. If Cancel status equals $00, the cancel function is disabled. If Cancel status equals $80, then cancel is enabled and nondestructive; if Cancel status equals $C0, cancel is enabled and destructive.

Status code: 15 (Escape)
Status list: Escape Mode status

Returns the status of escape mode (cursor command mode). If Escape Mode status equals $00, escape mode is disabled. If Escape Mode status equals $80, escape mode is enabled.

Status code: 16 (Cursor Position)
Status list: Horizontal byte , Vertical byte

Returns the current position of the cursor, relative to the upper left-hand corner of the viewport. Horizontal byte will range from Ø to the viewport width; Vertical byte will range from Ø to the viewport height.

Status code: 17 (Read Text Screen)
Status list: Character byte

Returns the ASCII value of the character at the current cursor position. Does not affect the screen or change the cursor position.

Status code: 18 (Preserve Contents of Viewport)
Status list: Viewport data

The contents of the viewport (both text and color information) are copied to the status list. The status list must be large enough to hold all of the data. In the 8Ø-column display modes, the amount of storage space required for the viewport data is equal to the width of the viewport multiplied by the height, plus 3. In the 4Ø-column modes, the space required is equal to twice the width of the viewport times the height, plus 3.

If you use this request under the assumption that you have already set the viewport, beware: if CONTROL-8 (Display Control Characters) was in effect when you tried to set the viewport, the viewport was not set (because the request was a control code).

Console Control Requests

The following list gives the control code and the contents of the control list for each console-driver control request. Note that the dollar sign ($) is used to indicate hexadecimal values.

Control code: Ø (Reset Console)
Control list: (nil)

Sets all console options to their default values.

Control code: 1 (Restore Status Table)
Control list: Buffer size, Buffer

Restores the status table of the console from the buffer. The buffer size and the buffer should have been filled with information by a previous Preserve Status Table status request; it is not advisable to enter your own values into the status table.

Control code: 2 (Termination Character)
Control list: Line status; Line character

Sets the line-termination status and character. If Line status is equal to $8Ø, then pressing the character whose ASCII code is specified in Line character will terminate an input request; if Line status equals $ØØ, only satisfying the proper character count will terminate an input.

Control code: 3 (Keyboard Mode)
Control list: Mode status

Sets the keyboard mode. If Mode status equals $ØØ, then a read request to the keyboard returns a single byte of ASCII code for each keystroke. If Mode status equals $8Ø, a read request returns a byte of ASCII code plus an additional byte of keyboard data. The format of the additional data byte is given in Appendix G.

Control code: 5 (Type-ahead Buffer size)
Control list: Buffer size

Sets the size of the type-ahead buffer, in keystrokes. If Buffer size = Ø, type-ahead is disabled.

Control code: 5 (Flush Typeahead Buffer)
Control list: (nil)

Removes all characters from the type-ahead buffer; performs the same function as pressing CONTROL-6 on the numeric keypad. This request has no effect if type-ahead is disabled or if the console has already emptied the type-ahead buffer into the input buffer.

Control code: 6 (Attention Event)
Control list: Priority, Event ID, Event-handler address, Attn char

Sets the current Attention Event parameters. The Attention event monitors keystroke data and tests for the particular character defined as the Attention character. When it detects the Attention character, the console driver flushes the type-ahead buffer and, if type-ahead is enabled, puts the character into the buffer. The console also turns off the event so that it cannot recur until the program reactivates it. Then the console passes control to the Event-handler address, as if it had executed a JSR instruction. If the Any-key event happens at the same time as the Attention event, the Any-key event takes precedence. Refer to the *Apple III SOS Reference Manual* for a complete description of events.

The Event-handler routine must be present throughout the period when it is possible for the event to be active. If your program cannot guarantee that the Event-handler routine will remain in memory, you should not use events.

Control code: 7 (Reserved)
Control list: (nil)

This request code is reserved for future assignment.

Control code: 8 (Any-Key Event)
Control list: Priority, Event ID, Event-handler address

Sets the current Any-Key Event parameters. The Any-Key Event is triggered by the pressing of any key. When the Any-key Event detects a character, the console driver flushes the type-ahead buffer and, if type-ahead is enabled, puts the character into the buffer. The console also turns off the event so that it cannot recur until the program reactivates it. Then the console passes control to the Event-handler address, as if it had executed a JSR instruction. If the Any-key event happens at the same time as the Attention event, the Any-key event takes precedence. Refer to the *Apple III Technical Reference Manual* for a complete description of Events.

The Event-handler routine must be present throughout the period when it is possible for the event to be active. If your program cannot guarantee that the Event-handler routine will remain in memory, you should not use events.

Control code: 9 (Reserved)
Control list: (nil)

This request code is reserved for future assignment.

Control code: 10 (No Wait Input)
Control list: No Wait status

Sets the status of the No Wait mode. If No Wait status equals $8Ø, then all input requests will return only characters that were in the type-ahead buffer at the time of the input request; the console will not wait until the user types the termination character or satisfies the character count. If No Wait status equals $ØØ, input is processed normally.

Control code: 11 (Screen Echo)
Control list: Screen Echo status

Sets the status of the Screen Echo mode. Only the two high-order bits are used. If you set Screen Echo status to $ØØ, then no keypresses will be echoed to the screen; in addition, there will be no cursor display, no ESCAPE mode, no character copy, and no cursor movement during backspace and cancel. If you set Screen Echo status to $8Ø, input characters will be echoed to the screen, and the other functions listed will all operate normally.

You can also control echoing of control characters. If Screen Echo status is set to $ØØ or $8Ø, as above, control characters typed on the keyboard will not be echoed to the screen, but if you set Screen Echo status to $CØ, then, in addition to normal character echoing, control characters will also be echoed to the screen.

Control code: 12 (Retype)
Control list: Retype status

Sets the status of Retype mode. If Retype status equals $ØØ, retype is disabled. If Retype status equals $8Ø, retype is enabled.

Control code: 13 (Backspace)
Control list: Backspace status

Sets the status of the backspace function. If Backspace status equals $00, the backspace function is disabled. If Backspace status equals $80, then backspace is enabled and non-destructive; if Backspace status equals $C0, backspace is enabled and destructive.

Control code: 14 (Cancel)
Control list: Cancel status

Sets the status of the cancel function. If Cancel status equals $00, the cancel function is disabled. If Cancel status equals $80, then cancel is enabled and non-destructive; if Cancel status equals $C0, cancel is enabled and destructive.

Control code: 15 (Escape)
Control list: Escape Mode status

Sets the status of escape mode (cursor command mode). If Escape Mode status equals $00, escape mode is disabled. If Escape Mode status equals $80, escape mode is enabled.

Control code: 16 (Download Character Set)
Control list: Character set

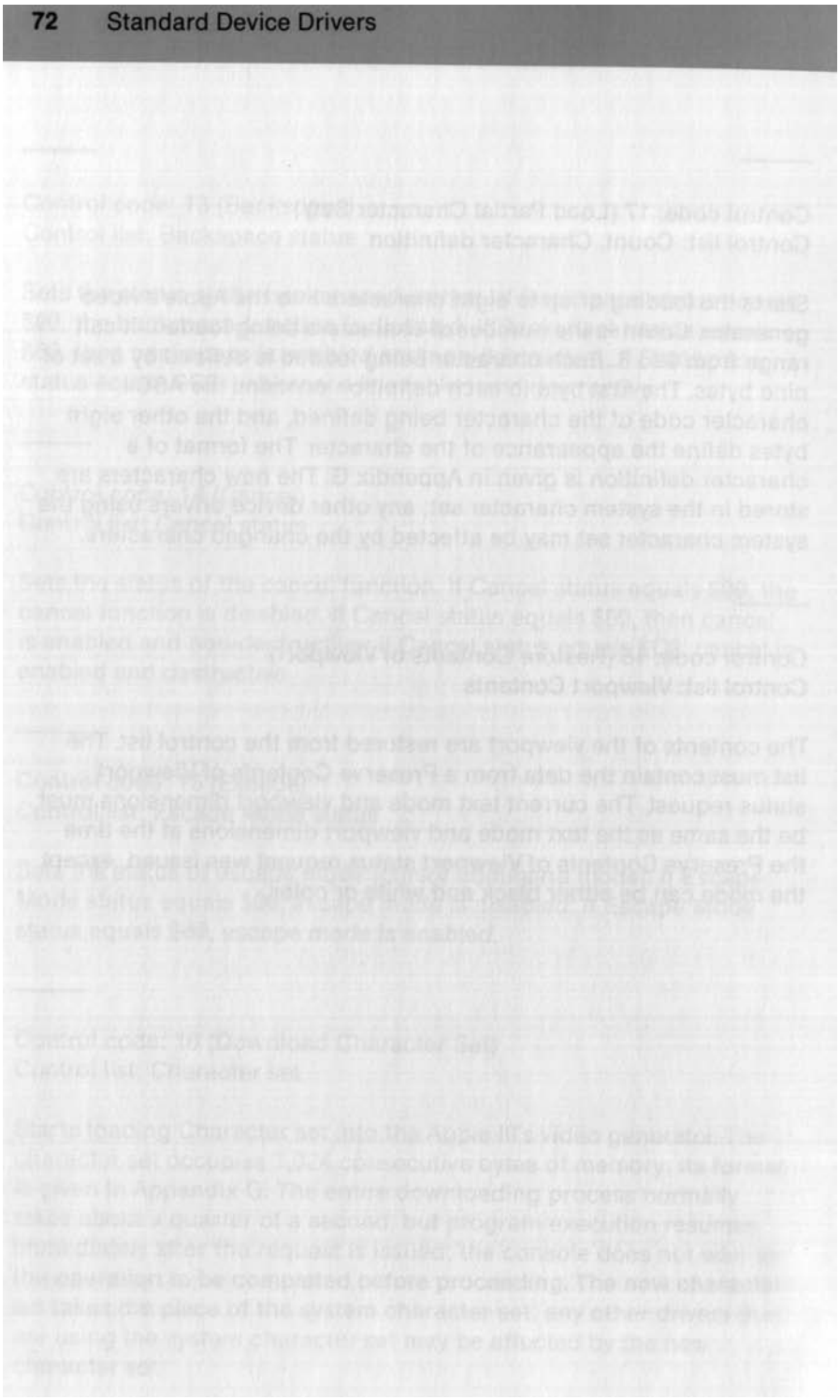
Starts loading Character set into the Apple III's video generator. The character set occupies 1,024 consecutive bytes of memory; its format is given in Appendix G. The entire downloading process normally takes about a quarter of a second, but program execution resumes immediately after the request is issued; the console does not wait for the operation to be completed before proceeding. The new character set takes the place of the system character set; any other drivers that are using the system character set may be affected by the new character set.

Control code: 17 (Load Partial Character Set)
Control list: Count, Character definition

Starts the loading of up to eight characters into the Apple's video generator. Count is the number of characters being loaded: it can range from Ø to 8. Each character being loaded is defined by a set of nine bytes. The first byte in each definition contains the ASCII character code of the character being defined, and the other eight bytes define the appearance of the character. The format of a character definition is given in Appendix G. The new characters are stored in the system character set; any other device drivers using the system character set may be affected by the changed characters.

Control code: 18 (Restore Contents of Viewport)
Control list: Viewport Contents

The contents of the viewport are restored from the control list. The list must contain the data from a Preserve Contents of Viewport status request. The current text mode and viewport dimensions must be the same as the text mode and viewport dimensions at the time the Preserve Contents of Viewport status request was issued, except the mode can be either black and white or color.



4

The Graphics Driver

In addition to the three text modes, which are controlled by the console driver, the Apple III's video generator can produce four modes of graphics on a video display. These four graphics modes are controlled by the .GRAFIX device driver.

Using the graphics driver, you can plot points, draw lines, write characters, and place blocks of predefined shapes on a graphics screen. You can also read the setting of any dot on a graphics screen.

The graphics device driver makes the graphics screen look like a text file. Opening the file .GRAFIX (with either an OPEN statement from BASIC or the REWRITE or RESET procedures from Pascal) turns on the graphics display and sets some default values inside the driver. Writing to the graphics driver will affect the appearance of the graphics screen; reading from the file will give you information about the appearance of the screen.

This method of communication with the driver, however, is inadequate for most high-level programming purposes. Although you can perform almost all operations on the graphics driver by sending and reading streams of characters, this becomes extremely inefficient when you attempt to do more interesting and complex things with graphics pictures. To help you, most high level languages have special applications packages that act as an interface between your high-level programs and the graphics driver. For example, Pascal uses the PGRAPH unit and Business BASIC uses the BGRAPH module. They allow you to express graphics commands in a high-level

style, similar to that of the programming language you are using. The commands are then converted into an output stream of characters and sent to the graphics driver. The operations performed by the driver are the same; only the means of invoking them has been simplified.

This chapter explains the features of the graphics driver that are available from all programming languages. It does not describe how to access those features from a given language; it only describes the driver's reaction to streams of output characters. To learn how to use the graphics driver from a high level language, refer to the documentation of the graphics application package for the high-level language.

The Graphics Modes

The Apple III's video display, like any television set, composes its pictures in a large rectangle of tiny dots. The size and quantity of these dots determines the *resolution* of the picture: the more dots there are in a picture, the finer (or higher) the resolution. The Apple III's video generator can create a picture four different ways, called *graphics modes*:

- 192 lines of 280 dots per line, black and white dots only;
- 192 lines of 280 dots per line, dots in any of 16 possible colors, with limitations;
- 192 lines of 560 dots per line, black and white dots only;
- 192 lines of 140 dots per line, dots in any of 16 possible colors, no limitations.

In addition, the Apple III can store two different pictures in each graphics mode. Thus you can display a picture in a certain graphics mode while creating another picture in the same mode, then switch the display to show the second picture. However, due to the different ways picture data is stored for the different graphics modes, it is difficult to display a picture in one graphics mode while creating another picture in a different mode.

Two kinds of manipulation can be performed on a picture: vector manipulations and block manipulations. All functions of the graphics driver perform in the same manner regardless of the current graphics mode, despite the differences in screen dimensions or color capabilities.

Vector manipulations involve two fundamental structures: the dot and the line. You can place a dot at any point on the screen or draw a line between any two points. Any picture that you can define as a set of dots and lines can be drawn using vector manipulations. You cannot, however, control the size of the dots or the width of the lines.

Block manipulations involve defining two-dimensional arrays of bits and transferring them as whole entities onto the screen. Block manipulations are used for placing predefined shapes (in rectangular blocks) at any position in the picture.

Graphics Tools

The graphics driver, like the console, maintains data about the boundaries of its screen, its current drawing position, and the colors in which it is drawing. These three pieces of data control the placement and appearance of images on the graphics screen:

- The *graphics viewport*. This is a defined area on the screen in which all changes to the screen take place. The graphics driver will alter only those areas of the screen inside the current viewport; by changing the size and position of the viewport, you can protect images you have already placed on other areas of the screen. The viewport may be of any size and at any position, as long as it fits on the current graphics screen.

- The *position* of the graphics *pen*. In effect, the pen draws on the graphics screen, and its position at any given time is called the pen position. All points, lines, blocks, and characters are drawn by the pen. The pen position has a large range: it can be anywhere on or off the screen, inside or outside the viewport. However, operations with the pen affect the screen only when the pen's position is inside the viewport.

- The *pen and fill colors*. The pen color is the color in which all dots, lines, blocks, and characters are drawn. The fill color is the color that fills the screen during a Clear Viewport operation, and is also the color that is filled in behind blocks and characters when you draw them on the screen.

The viewport is defined by four integers that specify the left, right, top, and bottom borders of the graphics viewport with respect to the resolution limits of the current graphics mode. The left and bottom borders cannot be less than zero; the right and top borders must be less than the number of dots in a screen row or column.

The pen position is specified relative to the graphics screen by two integers. The lower-left corner of the screen is pen position 0,0. The horizontal and vertical position of the pen are specified as 16-bit signed integers, with negative numbers in two's complement form. This is the number format used for integer variables by Apple Business BASIC and Apple Pascal.

The pen and fill colors can be chosen from a set of sixteen colors. Each color is specified by a number from 0 to 15 (that's hexadecimal values $00 to $0F; binary 0000 to 1111). The value of black is 0; the value of white is 15; and the fourteen values in between are each assigned to a color. Tables of the colors and their values appear both in this chapter and in the appendices. On a black and white monitor, the colors are displayed as shades of grey, with the lighter shades corresponding to the higher values.

You can change the viewport size and position, the pen position, the pen and fill colors, and most other values used by the graphics driver by sending special sequences of characters, called *graphics screen control codes*, to the driver. Details on the control of graphics output are given in the following sections of this chapter.

The Color Operator Table

In addition to enabling you to protect selected areas of the graphics screen by setting the viewport, the graphics driver also enables you to protect individual colored areas inside the viewport by using the *color operator table*. The color operator table determines which colors can overwrite other colors inside the viewport.

The color operator table is an array with 16 rows and 16 columns, one of each for each color. The rows represent the pen color and the columns represent the screen color. Normally the operator table is set up in this manner:

		Screen Color															
Pen Color		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Black	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	Magenta	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	Dk. Blue	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	Purple	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	Dk. Green	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	Grey 1	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	Md. Blue	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	Lt. Blue	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	Brown	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	Orange	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
10	Grey 2	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
11	Pink	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
12	Green	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	Yellow	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
14	Aqua	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
15	White	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

Each element at the intersection of a pen color row and a screen color column determines the color that will be drawn when you attempt to draw over an area of that screen color with a pen of that pen color. In the normal table, drawing with a certain pen color always produces that same pen color, regardless of the screen color at that location. If you change an element of the table, the graphics driver will change the way it draws colors on the screen according to the position and value of the changed element.

Suppose you were drawing road maps on the graphics screen and you had a green background, surface streets drawn in black, and elevated freeways drawn in pink. Now suppose you wanted to draw railroad tracks in yellow so that the tracks would always go above the green background and black surface streets, but always below the pink freeways. Before drawing the tracks in yellow, you would set the element in the color operator table where the pen color is yellow and screen color is pink to the value of the color pink, so that when you draw your yellow tracks across the pink freeway, a pink line is drawn that blends into the rest of the freeway.

Suppose now that you wanted to draw subway routes that do not appear under train tracks or freeways, but appear dark blue over the green background and light blue over surface streets. Before drawing the subway routes you would set the pen color to dark blue, and set the color operator table to this:

Pen color	Screen color	Resulting color
Dk. Blue	Green (background)	Dk. Blue
Dk. Blue	Black (streets)	Lt. Blue
Dk. Blue	Pink (freeways)	Pink
Dk. Blue	Yellow (tracks)	Yellow

The color operator table has many other uses, but this is the most common one: protecting colored items already on the screen from being overdrawn by any or all other colors.

The table acts on the fill color in exactly the same manner it acts on the pen color. The fill color is used only in character and block output, and in clearing the viewport.

The table is set up and changed by two screen control codes, discussed in a later section of this chapter.

The Transfer Modes

The transfer modes give you another degree of control over how colors are placed on the screen. After the color operator table determines which of the 16 colors to put at the pen position, the transfer mode determines *how* it is put there.

The transfer modes operate on color data as four-bit binary quantities, and perform binary logical operations on colors. Thus some of the color transfers you can obtain with different transfer modes are interesting but not really useful. When you use different transfer modes, combined with different settings of the color operator table, you can get intricate coloring algorithms that are very difficult to predict. When you're using colors, it's advisable to use the color operator table to protect colors on the screen, the transfer modes to obtain special effects, and both only when you really know what's going on.

The transfer modes tell the graphics driver to take the drawing color (from the operator table) and the current screen color and perform one of eight binary operations on the two, then draw a dot in the resulting color at the current pen position.

There are four basic transfer modes: STORE, OR, XOR, and BIC. In STORE mode, the drawing color is transferred directly to the screen, ignoring the color that's there already. In OR mode, the graphics driver performs the logical OR of the two colors and plots the resulting color. In XOR mode, an Exclusive-OR operation is performed, and in BIC (Bit Clear) mode, each bit in the drawing color that is on will force its corresponding bit in the current screen color off.

Each of these modes has a variant in which the drawing color is inverted, bit for bit, before the operation is performed; the inverse (NOT) of a color is the color whose value is equal to 15 minus the value of the original color. This gives a total of eight distinct transfer modes:

Transfer Mode	Resulting Color
STORE	draw color
OR	draw color OR screen color
XOR	draw color XOR screen color
BIC	draw color BIC screen color
INVERSE STORE	(NOT draw color)
INVERSE OR	(NOT draw color) OR screen color
INVERSE XOR	(NOT draw color) XOR screen color
INVERSE BIC	(NOT draw color) BIC screen color

These modes, especially the BIC and inverse modes, are most useful in the black-and-white graphics modes. The screen color will always be either black (all zeros) or white (all ones), so if the drawing color is also all zeros or all ones, these binary operations will perform as you would think they should: as binary logical operators on each dot.

Tables of the resulting colors for each of these operations appear in Appendix B of this manual.

Graphics Output

After you open the .GRAFIX driver for output, you can use any file-oriented output command (such as BASIC's PRINT# and OUTPUT# statements or Pascal's WRITE and WRITELN) to send characters to the graphics driver.

Pascal traps certain control characters in WRITE and WRITELN statements. Certain graphics arguments can have the same values as those control characters (16 or greater), so you cannot use WRITE or WRITELN to send them to the driver. Use UNITWRITE instead.

The graphics driver recognizes 160 distinct characters. If a character received by the graphics driver is an ASCII control character (with a value from 0 to 31), the driver performs a special control function. These control characters, called *screen control codes*, are described in the next section.

If the character is a standard printing character (ASCII code 32 to 127), the driver will use the current graphics character set to place the image of that character on the current graphics screen. The character is placed below and to the right of the pen position and the pen is advanced to the right a distance equal to the width of the character. Any portions of the character that are not inside the viewport will not be drawn.

The graphics driver normally uses the system character set loaded with the System Configuration Program; this is the same default character set used by the console driver. Although the two drivers use the same default character set, the graphics driver can use an alternate character set. If you change the graphics character set, the console's character set is unaffected; but if you change the console character set and the graphics driver is using the system character set, the graphics character set will be changed along with the console's. Any change in the graphics character set will not, however, affect graphics characters already on the screen.

In addition to the 96 standard printing characters, the character set allows you to define 32 additional characters, one for each ASCII control character. If the character received by the graphics driver is an ASCII control character with its high bit set (giving it a value between 128 and 159), the graphics driver will clear the high bit and handle the character as a special printing character, placing its associated special character image on the graphics screen.

If the graphics driver receives a standard printing character with its high bit set (that is, a value between 160 and 255), the graphics driver will place a character on the screen. If the driver is using the standard character set, the character will be displayed as the standard character with the high bit off. If, however, the driver is using an alternate character set, the values 160 through 255 can be assigned to 96 additional characters. With the 96 standard characters and the 32 special printing characters mentioned above, this gives the Apple III a total of 224 distinct printing characters.

Screen Control Codes

The ASCII control characters with values between Ø and 31 are recognized by the graphics driver as graphics screen control codes. When the driver receives one of these codes during output, it performs a control function that changes the appearance or behavior of the graphics screen.

Some of the screen control codes require one or more arguments. These are extra characters immediately following the control code that do not perform their regular function, but instead supply extra information to a function invoked by the code. For example, the PEN COLOR control code (code 19) requires one argument whose value determines the color that is used to display characters. A segment of a program to set the pen color to dark blue (color 2) might be written in BASIC in this manner:

```
2Ø OPEN#1, ".GRAFIX"
3Ø SETPEN=19 : DARKBLUE=2
4Ø PRINT#1;CHR$(SETPEN);CHR$(DARKBLUE);
```

Because Pascal traps certain control characters, you must use the UNITREAD procedure to send characters whose values could be the same as the control characters. A Pascal example to set the pen color to color code 2 might read:

```
var: g_array: packed array [Ø..2Ø] of Ø..255;
begin
g_array[Ø]:=19; g_array[1]:=2;
unitwrite(3,g_array,2,,12);
end;
```

Eleven screen control codes require arguments; the number of arguments required by a screen control code range from one to as many as 2Ø.

Some control codes require 16-bit signed integers as arguments. Two characters are needed to send such an integer to the graphics driver. In Apple Business BASIC it is difficult to send such an integer directly to the graphics output device: you must first divide the number into its component bytes, making sure to keep the integrity of the two's

complement form for negative numbers. Apple recommends using BGRAPH, the graphics application package for BASIC, when you need to use control codes that require these arguments. If you must send the integers directly, this BASIC statement will convert an integer value in the variable VAL% into a two-character string in the variable VAL$ that can then be used as an argument of a control code:

```
1000 VAL$=CHR$(TEN(LEFT$(HEX$(VAL%),2))) +
     CHR$(TEN(RIGHT$(HEX$(VAL%),2)))
```

This statement will enable your BASIC programs to convert any integer between −32767 and +32767 into an argument of a screen control code.

Many control characters have no function in the graphics driver; such characters are ignored by the driver. The following is a list of the 18 functional screen control codes, their arguments, and their functions:

RESET VIEWPORT
Code 01, "SOH" (Start of Header)

Resets the viewport to the size of the current graphics screen. Does not affect the pen position.

SET VIEWPORT
Code 02, "STX" (Start of Text)

Sets the coordinates of the corners of the viewport. The eight argument bytes comprise four 16-bit signed integers:

- Left boundary (2 bytes)
- Right boundary (2 bytes)
- Bottom boundary (2 bytes)
- Top boundary (2 bytes)

All boundaries are relative to the lower-left corner of the screen, which is absolute position (0,0). If any boundary value is outside the screen limits for the current graphics mode, its value will be adjusted to fit the current graphics screen.

CHARACTER SET
Code Ø3, "ETX" (End of Text)

Sets the address and character size of the new character set to be used for placing characters on the graphics screen. Its arguments are:

Character set address (3 byte address)
Character width (1 byte integer)
Character height (1 byte integer)

As it is difficult for a user program to obtain three-byte memory addresses of items in the system's memory, Apple recommends that you use the graphics applications package for your language to perform this function. See the documentation for the language for details.

DRAWBLOCK
Code Ø4, "EOT" (End of Transmission)

Draws a predefined shape on the screen. It takes a block of bits and transfers them to the the current graphics screen, placing them below and to the right of the current pen position. Each one bit in the block will cause the pen color to be drawn on the screen; each zero bit will cause the fill color to be drawn on the screen. Both the pen and fill colors are passed through the color operator table and the transfer mode before being placed on the screen. The arguments are:

Block address (3 byte address)
Bytes per row (2 byte integer)
Column bit displacement (2 byte integer)
Row displacement (2 byte integer)
Pixel width of block (2 byte integer)
Pixel height of block (2 byte integer)

Column displacement is measured from the right edge of the screen; row displacement is measured from the top.

Once again, this function involves the handling of two-byte memory addresses, and Apple recommends that you use the graphics applications package for your language to perform this function. See the documentation for the language for details.

PEN LINE FEED
Code 1Ø, "LF" (Line Feed)

Moves the pen vertically toward the origin an amount equal to the dot height of the current character set. This is the equivalent of a line feed in a text mode. The pen is free to move out of the viewport: no scrolling is performed.

PEN RETURN
Code 13, "CR" (Carriage Return)

Moves the pen horizontally to the left edge of the viewport. No vertical pen movement occurs. This is the equivalent of a carriage return in a text mode.

TURN OFF SCREEN
Code 14, "SO" (Shift Out)

Disables the Apple III's video generator. The video monitor will show a blank screen regardless of the information on any text or graphics screen. The information on the graphics screen is not lost; the image will be restored to the screen when a TURN SCREEN ON command is issued (see below). When the screen is off, the Apple III's central processor operates slightly faster. This command has no effect if the screen is already off.

TURN SCREEN ON
Code 15, "SI" (Shift In)

Enables the Apple III's video generator. The video monitor will display graphics in the currently selected graphics mode. If one of the text modes was in effect, sending the TURN SCREEN ON command to the graphics driver will return the display to the graphics mode. This command has no effect if the current graphics screen is already being displayed.

GRAPHICS MODE
CODE 16, "DLE" (DATA LINK ESCAPE)

Sets the mode used by the graphics driver to produce the graphics display. The next character following the code specifies the graphics mode to set. These are the argument characters and the graphics modes they specify:

Mode		Character
Ø	28Ø by 192, black-and-white	Ø
1	28Ø by 192, limited color	1
2	56Ø by 192, black-and-white	2
3	14Ø by 192, full color	3
4	Mode Ø, alternate screen	4
5	Mode 1, alternate screen	5
6	Mode 2, alternate screen	6
7	Mode 3, alternate screen	7

Only the three least significant bits of the character are used; the others are ignored.

The GRAPHICS MODE command takes effect immediately, but the chosen mode will be displayed only when a TURN SCREEN ON code (code 15) is sent to the graphics driver. Any commands sent to the driver after a GRAPHICS MODE command will always be processed in the new mode.

The GRAPHICS MODE command adjusts the setting of the viewport only when the viewport will not fit on the new graphics screen. When changing from one mode to another mode that has smaller screen dimensions, the left and right boundaries of the viewport will be adjusted so that they fit into the new mode. When changing from one mode to another mode with larger screen dimensions, the viewport remains its smaller size. It is advisable to make sure the viewport setting is correct after changing graphics modes.

This command does not change the information stored on the screen, but it does change the way the Apple III interprets that information. Changing modes with information already on the screen will not modify that information to make it appear the

same in the new mode. For example, when changing from mode Ø to mode 1, though the arrangement of dots will appear the same, the color added by the new mode will alter the picture in unusual ways. Thus it is advisable to clear the screen after selecting a graphics mode.

RESET COLOR OPERATOR TABLE
Code 17, "DC1" (Device Control 1)

This function sets the color operator table to its default value. Until the operator table is changed again, it will allow any color to overwrite any other color in the viewport; no color has priority.

SET COLOR OPERATOR TABLE
Code 18, "DC2" (Device Control 2)

This function allows you to change any number of elements in the color operator table. It has five bytes of arguments:

Drawing Colors (2 byte binary mask)
Screen Colors (2 byte binary mask)
Result Color (1 byte color value)

The first two arguments specify which drawing colors (for pen color or fill color) are being affected by the change. Each bit position in the first two-byte value indicates one drawing color: the most significant bit in the first byte is color 15, white; the least significant bit in the second byte is color Ø, black. Each bit that is a 1 indicates that its associated color is being affected.

The next two arguments form another two-byte binary mask, selecting screen colors to be changed. The bits are arranged in the manner described above. Every element in the color operator table that is in both a selected pen color and a selected screen color will be changed.

The final argument specifies the color that is to be drawn when the selected pen (or fill) colors are used to draw over the selected screen colors. This should be a standard color value, from Ø to 15.

For example, if you want to change the operator table so that when either of the greys (colors 5 and 10) is drawn over any of the blues (colors 2, 6, and 7) the resulting color should be orange (color 9), these five bytes should be passed in order as arguments to the SET OPERATOR TABLE code:

Argument Number	Decimal Value	Binary Value	Function
1	8	00000100	Drawing colors 10
2	32	00100000	and 5
3	0	00000000	Screen colors
4	200	11000100	7, 6, and 2
5	9	00001001	Result value 9

Once you set an element in the color operator table, it remains set that way until you reset it individually, reset the entire table, or close and reopen the graphics driver.

PEN COLOR

Code 19, "DC3" (Device Control 3)

Sets the pen color to the value specified by the next character. All subsequent lines, dots, blocks, and characters will be drawn in the new color. In the black and white modes, any non-black color appears white. Only the lower four bits of the argument are used.

Here are the sixteen colors, their values, and argument characters commonly used to specify them:

Color	Value	Character	Color	Value	Character
Black	0	0	Brown	8	8
Magenta	1	1	Orange	9	9
Dark Blue	2	2	Grey	10	:
Lavender	3	3	Pink	11	;
Dark Green	4	4	Green	12	<
Grey	5	5	Yellow	13	=
Medium Blue	6	6	Aqua	14	>
Light Blue	7	7	White	15	?

FILL COLOR

Code 20, "DC4" (Device Control 4)

Sets the fill color to the value specified by the next character. Any subsequent clear viewport commands will clear the viewport to the fill color. All characters placed on the graphics screen and all blocks placed by DRAWBLOCK will be placed on a background of the fill color. In the black and white modes, any non-black fill color appears white. Only the lower four bits of the color specification character are used. The colors and their associated characters are given in the table above.

TRANSFER MODE

Code 21, "NAK" (negative Acknowledge)

Sets the graphics transfer mode to the value specified by the argument character. There are eight transfer modes:

Mode	Character
STORE	0
OR	1
XOR	2
BIC	3
INVERSE STORE	4
INVERSE OR	5
INVERSE XOR	6
INVERSE BIC	7

Only the lower three bits of the argument character are used; the others are ignored. The transfer mode affects all items placed on the screen, and operates on the drawing color (from the color operator table) and the current screen color for each point. Logical tables defining the OR, XOR, and BIC modes for all color combinations are given in Appendix B.

DRAW LINE

Code 24, "CAN" (Cancel)

Draws a line in the current pen color from the current pen position to the position specified by the arguments. The arguments are:

New X-position (2 byte integer)
New Y-position (2 byte integer)

The pen position remains at the new position. Any portions of the line that are not inside the viewport will not be drawn, but the pen position will change nonetheless. The appearance of the line is controlled by the pen color, the color operator table, and the transfer mode (in that order).

PLOT POINT
Code 25, "EM" (End of Medium)

Moves the pen to the given position and plots a single point. The pen remains at the new position. The arguments are the X-position and Y-position of the point to plot, in the same format given for DRAW LINE.

If the point is not within the current viewport, it will not be drawn. The appearance of the point is controlled by the pen color, the color operator table, and the transfer mode.

MOVE PEN
Code 26, "SUB" (Substitute)

Moves the pen to the given position. The screen is not affected. The arguments are the new X-position and new Y-position, as above.

CLEAR VIEWPORT
Code 28, "FS" (Field Separator)

Clears the contents of the entire viewport to the current fill color, modified by the color operator table and the transfer mode. The pen position is not changed.

The Limited Color Mode

Mode 1 and its alternate screen, Mode 5, were described earlier as having limited color capabilities. This section describes those capabilities in detail. The limited color mode has both advantages and drawbacks: it has the highest resolution of all the color displays on the Apple III, but there are restrictions on where and how colors can be placed on the screen.

You can think of the limited-color screen in two parts: a 280 by 192 black-and-white display, and a 40 by 192 color overlay:

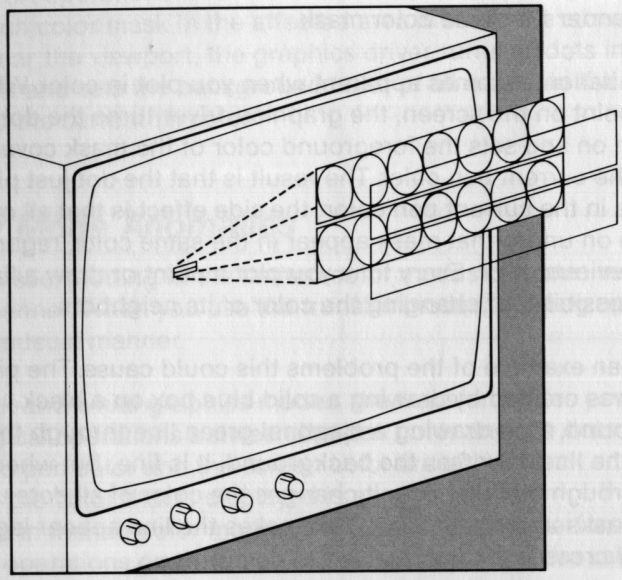


Figure 2. Limited Color Mode

The black-and-white display is just like the 280 by 192 black-and-white display of Mode 0 and Mode 4. Each individual dot can be on or off.

The color overlay gives color to the image produced on the black-and-white display. Each line on the color overlay is made up of 40 color masks, and each mask covers a short row of seven adjacent dots on the black-and-white display. Each color mask has two colors associated with it: a foreground color and a background color. Any dot that is on will be displayed in the forground color of the mask covering that dot; any dot that is off will be displayed in the background color of the mask covering that dot.

For example, if a certain mask has been assigned a foreground color of pink and a background color of orange, and four dots under that mask are turned on, then those four dots will be displayed as pink and the other three dots will be displayed as orange.

This is the fundamental limitation of the limited color mode: the group of seven dots under a color mask can be displayed in only two colors at a time. It is not possible to have dots in more than two colors under the same color mask.

This limitation becomes apparent when you plot in color. When you plot a point on the screen, the graphics driver turns the dot at that position on and sets the foreground color of the mask covering that dot to the current pen color. The result is that the dot just plotted appears in the current pen color; the side effect is that all other dots that are on under that mask appear in the same color, regardless of their previous color. Every time you plot a point or draw a line, there is the possibility of changing the color of its neighbors.

Here is an example of the problems this could cause. The picture below was created by drawing a solid blue box on a black background, then drawing a diagonal green line through the box. Where the line traverses the background, it is fine; but where the line goes through the blue box, it changes the color of all dots under the color masks used by the line. This makes the line appear jagged where it crosses the box.

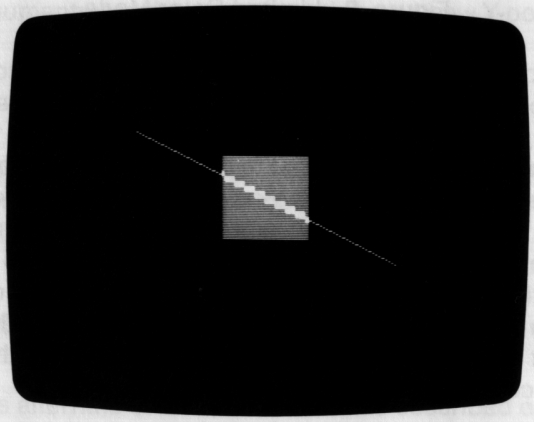


Photo 9. Color Anomalies

The background color of a color mask is changed when the graphics driver clears the viewport and during text placement and DRAWBLOCK operations. In all three operations, the background color of each color mask in the affected area is set to the current fill color. To clear the viewport, the graphics driver turns all dots in the viewport off and sets the background color of all masks in the viewport to the current fill color.

Transfer Mode Anomalies

As you can see, plotting in the limited color modes operates in an unusual manner. When you use the transfer modes, plotting operates in a *very* unusual manner.

In the black-and-white graphics modes, the transfer modes perform one-bit logical operations between the pen color and the dots on the screen. The pen color and the screen color are either black or non-black, and the color that is displayed is a logical function of these two values. In the full-color modes, the transfer modes perform four-bit logical operations on the value of the pen color and the value of the screen color, and the resulting value is the color that gets displayed. In the limited color modes, *both* of these operations take place.

For example, if you set the transfer mode to OR and plot a point, two things happen. First, the pen color (black or non-black) is logically OR-ed with the state (off or on) of the dot at the point being plotted, and the dot is then turned on or off according to the result of the logical operation. Then the four-bit value of the pen color is logically OR-ed with the value of the foreground color of the color mask covering that dot, and the four-bit result of that operation becomes the new foreground color of that color mask.

The results are always logically predictable, but the process can lead to coloring paradigms of byzantine complexity. If you decide to use transfer modes in the limited color modes, you should have a clear mind, a thorough grasp of the concepts involved, and a boundless enthusiasm for the intricate and beautiful results you can obtain.

Memory Requirements

The different graphics display modes use different amounts of main memory:

Mode Numbers	Memory Used	Description
Ø and 4	8K bytes	28Ø by 192 black-and-white
1 and 5	16K bytes	28Ø by 192 limited color
2 and 6	16K bytes	56Ø by 192 black-and-white
3 and 7	16K bytes	14Ø by 192 full color

Most languages on the Apple III allow any memory space not used by the graphics modes to be used for program or data storage. The graphics driver does nothing to prevent this, but if you do this, you must make sure that you do not put your program or data where the driver will write graphics data.

Reading the Screen

Performing a BASIC INPUT# statement or a Pascal READ statement from the file .GRAFIX will return a single character or string of characters. The value of the first character returned represents the screen color of the current pen position. Returned values will be alphabetic characters with values from 64 to 79; the true color value can be obtained by subtracting 64 from the value of the first character in the returned string. If the graphics mode is black-and-white, the values 64 (for black) or 79 (for white) are returned.

A color value is returned regardless of whether the pen is in the viewport. If, however, the pen position is not inside the boundaries of the current graphics screen, a character with the value of 128 will be returned.

The Graphics Configuration Block

The chapter on the System Configuration Program describes how you edit driver parameters. One of the items you can choose to edit with this option is the Configuration Block data. When you select this item in the .GRAFIX driver, you see a display similar to this. (The columns are numbered in hexadecimal; if you replace the X in one of the numbers down the left side with the column number of a byte you are interested in, you get the hexadecimal value of the location of that byte in the Configuration Block.)

SYSTEM CONFIGURATION PROGRAM—
EDIT DRIVER CONFIGURATION BLOCK

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Øx—	[ØØ]	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	17	Ø1	ØØ	ØØ	BF	ØØ	ØF	ØØ
1x—	ØØ	ØC	ØØ	Ø7	Ø8	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	ØØ	11	11
2x—	11	11	11	11	11	11	22	22	22	22	22	22	22	22	33	33
3x—	33	33	33	33	33	33	44	44	44	44	44	44	44	44	55	55
4x—	55	55	55	55	55	55	66	66	66	66	66	66	66	66	77	77
5x—	77	77	77	77	77	77	88	88	88	88	88	88	88	88	99	99
6x—	99	99	99	99	99	99	AA	AA	AA	AA	AA	AA	AA	AA	BB	BB
7x—	BB	BB	BB	BB	BB	BB	CC	CC	CC	CC	CC	CC	CC	CC	DD	DD
8x—	DD	DD	DD	DD	DD	DD	EE	EE	EE	EE	EE	EE	EE	EE	FF	FF
9x—	FF	FF	FF	FF	FF	FF										

This configuration block data specifies the initial default conditions for graphics each time the .GRAFIX driver is opened for use. You can change any of these initial conditions by editing the configuration block. The conditions set by each of the displayed configuration block bytes is as follows:

Byte# (hex)	Default Condition Set by This Byte
ØØ	Graphics Mode (Ø to 7). Only the low-order three bits are used.
Ø1	Transfer Option (Ø to 7).
Ø2	Cursor x-position: low byte.
Ø3	: high byte.
Ø4	Cursor y-position: low byte.
Ø5	: high byte.
Ø6	Viewport left edge: low byte.
Ø7	: high byte.
Ø8	Viewport right edge: low byte.
Ø9	: high byte.
ØA	Viewport bottom edge: low byte.
ØB	: high byte.
ØC	Viewport top edge: low byte.
ØD	: high byte.
ØE	Pen color (Ø to F, hexadecimal)
ØF	Fill color (Ø to F, hexadecimal)
1Ø	Character font address: low byte.
11	: high byte.
12	: extend byte.
13	Character cell bit width.
14	Character cell bit height.
15	(Not currently used.)
16	Color operator table, first two result colors in the first row. The four high bits are the result color for source color Ø and screen color Ø, and the four low bits are the result color for source color Ø and screen color 1.
17	Color operator table, second two result colors in first row. See Appendix B, Graphics Quick Reference, for a diagram showing the rows and columns of the color operator table.
18-1D	Color operator table, result colors in remainder of first row.
1E-25	Color operator table, result colors in second row.
26-2D	Color operator table, result colors in third row.
2E-95	Color operator table, result colors in remaining rows.

Advanced Techniques

Whenever you send a control character to the console driver, the language system you are using actually transmits the character by means of a system call. System calls are the normal method that assembly-language programs (including interpreters and language systems) use for communicating with files and devices on the Apple III.

Instead of sending control characters as part of the output character stream, you can control the operation of the graphics driver by issuing your own system calls. The particular system calls that you use to control device drivers are the D_STATUS and D_CONTROL calls.

Interpreters (such as BASIC) and language systems (such as Pascal) provide more convenient ways of communicating with files and devices, so that user programs don't ordinarily need to resort to system calls. However, some language systems on the Apple III provide ways for you to make SOS calls directly to the system. For example, Apple III Pascal includes a procedure called UNITSTATUS that is used for both D_STATUS and D_CONTROL system calls. The use of this procedure is described in Appendix H.

These system calls require three parameters, like this:

```
D_STATUS (device number, status code, status list)
D_CONTROL (device number, control code, control list)
```

where device number specifies the device you want to get status information from or send control information to; status code and control code specify the particular information to get or to send; and status list and control list specify the location in the program's memory space where the information is or will be put. For more details on using system calls from assembly language programs, refer to the Apple III SOS Reference manual.

Graphics Status Requests

The following list gives the status code and the contents of the status list for each graphics driver status request.

Status code: 1 (Preserve Status Table)
Status list: Buffer size, Buffer
Copies the status table of the graphics driver into the buffer portion of the statuslist. The status table contains all of the current driver parameters; it holds the current graphics mode, transfer mode, pen position, viewport boundaries, pen and fill colors, character set address, and color operator table. Once the status table is preserved, you can change the graphics status as much as you like and still be able to restore its original status by issuing a Restore Status Table control request.

You must set the buffer-size byte to the number of bytes available in the status list (in this case, 15Ø) before executing the request; the graphics driver will place the aØual number of bytes used into Buffer size and store the status information into the buffer. The internal arrangement of this list is for the driver's use only; you should not attempt to do anything other than preserve and restore the status table.

Graphics Control Requests

The following list gives the control code and the contents of the control list for each graphics driver control request.

Control code: Ø (Reset Console)
Control list: (nil)

Sets all graphics options to the default values specified in the configuration block of the .GRAFIX driver.

Control code: 1 (Restore Status Table)
Control list: Buffer size, Buffer

Restores the status table of the graphics driver from the buffer. The buffer should have been filled with information by a previous Preserve Status Table status request; it is not advisable to enter your own values into the status table.

5

The Printer Driver

The standard device driver .PRINTER enables your programs to send output to a letter-quality serial printer, such as the QUME Sprint 5. The .PRINTER driver is an output-only driver using the Apple III's built-in RS-232-C serial interface; it will not allow you to read information through the RS-232-C port. For that, you must use the driver .RS232 described in Chapter 6.

The printer driver is configured to drive the QUME Sprint 5. To use another serial printer, you should first examine the configuration values given below. If changes are necessary, use the System Configuration Program to change the printer driver's configuration block. An explanation of this procedure appears in the section "Changing the Configuration Block" later in this chapter.

Basic Operations

Your Apple Business BASIC programs can send characters to the .PRINTER driver like this:

```
110 OPEN#1, ".PRINTER"
120 OUTPUT#1
130 PRINT "This is a test."
140 CLOSE#1
```

Apple Pascal programs can use the .PRINTER driver this way:

```
f:text;
begin
  rewrite(f,'.PRINTER');
  writeln(f,'This is a test.');
  close(f);
end;
```

When you open the file .PRINTER for output, the printer driver is initialized. After the file has been opened, each PRINT# or OUTPUT# statement from BASIC or WRITE or WRITELN statement from Pascal puts the text into an output buffer. From there, the .PRINTER driver transfers the data to its own buffer, then it returns control to the program that sent the output.

The printer driver then proceeds to send characters, one at a time, to the RS-232-C serial output port, to be accepted by a printer connected to that port. This process continues until all text in the buffer has been output, or the system is re-booted, or the computer is turned off.

Instead of waiting until the printing is finished, the program that requested output continues executing while the printing is being done. The program executes while the driver is waiting for the printer to print the characters it has just sent.

If for some reason the printer is unable to accept characters, the printer driver waits until it is ready. When the printer is ready to go again, output resumes from the same place it was stopped, just as though the printer had never stopped at all.

Printer Output

Opening the file .PRINTER for output resets only the printer driver; no initialization of the printer itself is done. Sending characters to the open file will cause those characters to be queued for output. As soon as the printer is ready to accept the characters, the characters will be sent to the printer to be printed on the page.

The .PRINTER driver communicates with the printer via a hardware handshake. It monitors the Data Set Ready and Data Carrier Detect signals (pins 6 and 8, respectively, on the serial port) to verify that the printer is ready. If for some reason the printer is unable to accept characters, it turns off one or both of these signals, causing the driver to wait. When the printer is ready to go again, it turns both signals on, and the driver resumes output as if the printer has never stopped.

If the printer is connected to the Apple~III through a modem eliminator (see the section CONNECTING THE PRINTER), it should provide its handshake signals on Data Terminal Ready and Request To Send (pins 2Ø and 4, respectively). If you are using a Qume printer, its internal switch should be in the No-modem position.

Closing the printer file has no effect on the characters already sent to the printer but as yet unprinted; the driver waits until all characters have been transmitted. The only way to flush the printer output buffer is by turning off the computer or re-booting the system.

The printer driver will accept all 128 ASCII characters, with or without their high bits set. It neither interprets nor modifies the stream of output characters: the characters you send to the printer driver are passed on directly to the serial output port. The output port can truncate the output characters according to the output format that has been set; see the next section for details.

Usually, the operation of a printer is controlled by sending control characters in the output data stream. Thus, control characters are passed through to the printer without being intercepted by printer driver.

Changing the Configuration Block

The printer driver's configuration block contains five parameters that control the way it communicates with a printer. The first two parameters control the data rate and the format of the output data sent to the printer. The last three parameters control the amount of time the printer driver waits after it sends a line feed, carriage return,

or form feed character. These parameters are normally set to default values appropriate for a Qume Sprint 5 printer at 1200 baud.

The first two parameters, *data rate* and *data format*, are vital to the communication between the printer driver and the printer. If they are not set correctly, the printer will not print characters correctly.

The other three parameters enable the driver to accommodate the mechanism of the printer. Most printers take a certain amount of time to move the printing carriage and the paper platen, so these three parameters are used to tell the printer driver to wait long enough for the printer to perform these time-consuming movements.

To examine or change the values of the configuration parameters, use the System Configuration Program to load the current system configuration from one of your boot diskettes, then use the EDIT DRIVER PARAMETERS option to edit the .PRINTER driver. Instructions for this procedure are given in Chapter 2, The System Configuration Program.

From the device driver's parameter display, choose the option EDIT CONFIGURATION BLOCK. The screen will display the printer's configuration block:

Photo 10. Printer Driver Configuration Block

The values of the parameter bytes appear in the upper-left corner of the display.

Byte number 1 is the data rate. Its value (shown in hexadecimal) determines the speed at which the Apple III communicates with the printer. There are nine possible speeds, specified in Baud, which is the unit of measure equal to the total number of bits transmitted per second, including start, stop, and parity bits.

Value	Speed	
Ø3	11Ø	baud (Teletypewriter speed)
Ø4	134.5	baud
Ø6	3ØØ	baud (A common telecommunications speed)
Ø7	6ØØ	baud
Ø8	12ØØ	baud (A common printer speed)
Ø9	18ØØ	baud
ØA	24ØØ	baud
ØC	48ØØ	baud
ØE	96ØØ	baud

The normal value is Ø8, for 12ØØ baud. To change the speed, use the four arrow keys to position the indicator (a pair of square brackets) on the first value in the configuration block. Type the two characters in the new speed value and press RETURN. The value will change and the indicator will move to the next value.

The second byte sets the data format. Its value determines the arrangement of bits in the data stream that the computer sends to the printer. There are nine standard formats:

Value	Format
22	7 bits, odd parity (odd number of 1's)
26	7 bits, even parity (even number of 1's)
2A	7 bits, mark parity (parity bit always 1)
2E	7 bits, space parity (parity bit always Ø)
ØØ	8 bits, no parity
42	6 bits, odd parity (odd number of 1's)
46	6 bits, even parity (even number of 1's)
4A	6 bits, mark parity (parity bit always 1)
4E	6 bits, space parity (parity bit always Ø)

However, if you select 11Ø baud by setting byte number 1 to the value Ø3, the driver will force two stop bits regardless of the value of the second byte.

The normal value of byte number 2 for the Qume printer is 22: seven data bits transmitted per character, with the parity (high) bit a 1 if the number of data bits set to 1 is odd. (The parity bit is intended to be used for error checking.)

The next three parameters control the amount of time the driver waits for the printer to do slow mechanical operations. These times are specified in equivalent character times at the specified data rate. The total number of bits per character, including non-data bits, is usually 1Ø or 11, so the approximate character rate can be found by dividing the baud rate by 1Ø.

The third byte sets the number of character times the printer driver waits after it sends a carriage return character (ASCII CR, value 13) to the printer. It allows the printer this much time to return the printing element to the left margin. The normal delay value is zero, which produces no delay. The delay value can range from Ø to 255 and must be specified in hexadecimal. (A decimal-to-hexadecimal conversion table is supplied in Appendice J.)

The fourth byte sets the number of character times the printer driver waits after it sends a line feed character (ASCII LF, value 1Ø) to the printer. This gives the printer the time it needs to advance the paper one line. The normal delay value is zero, which produces no delay. The delay value can range from Ø to 255 and must be specified in hexadecimal.

The fifth sets the number of character times the printer driver waits after it sends a form feed character (ASCII FF, value 12) to the printer. This gives the printer the time it needs to advance the paper to the top of the next printing page. The normal delay value is zero, which produces no delay. The delay value can range from Ø to 255 and must be specified in hexadecimal.

Once you have changed all the values you wish to change, return to the Configuration menu and choose the GENERATE NEW SYSTEM

option to save the newly modified printer driver on your boot diskette, as described in Chapter 2.

Connecting the Printer

The standard method of connecting two devices that communicate via the RS-232-C standard is by a cable with a 25-pin connector. The same type of connector is used at both the computer and the printer, but simply connecting them together won't work. If you do this, the pin the printer receives data on will be connected to the pin the computer receives data on, and similarly for the pins each transmits data on, and nothing will happen.

The convention adopted by the EIA (Electronic Industries Association) calls for both the printer and the computer to have the same pin connections, because each of them must be capable of being connected to the same type of data communications equipment, such as a *modem* (modulator/demodulator, used for data transmission over telephone lines).

To connect a modem to your Apple III, you simply plug its 25-pin connector into the RS-232 port, but to connect a printer, you must use the modem eliminator supplied with the Apple III. The modem eliminator is a short cable with a 25-pin connector on each end.

Even though the standard 25-pin plug on the printer cable will fit the RS-232 port on the Apple III, the printer will not work when connected this way. The Apple III and the printer must each connect either to a modem (or other piece of data communications equipment) or to a modem eliminator.

6

The RS232 Driver

Introduction

The .RS232 driver makes it possible for the Apple III to exchange information with a large variety of devices that use the RS-232-C serial interface. With it the Apple III can talk to another Apple III, an Apple II, a terminal, a printer, or a remote device connected through a modem.

The chief advantage of this driver over the .PRINTER driver is that it can perform two-way communication. Using this driver, application programs can:

- communicate with devices other than printers (terminals, computers, etc.)
- communicate with remote printers, terminals and computers, etc., via a modem
- connect the Apple III with an ever-growing variety of data banks and information services
- emulate many types of terminals

- change characteristics of the RS-232-C port using software commands rather than physical switches

Other advantages of the .RS232 driver are:

- it can be tailored to a greater variety of printers than .PRINTER (though .RS232 occupies more space)
- it can use any one of several signalling procedures (called "protocols") to control communications

In the discussion that follows, all numbers are hexadecimal unless otherwise noted. Hexadecimal numbers are marked in most cases by a dollar sign prefix, but you should not actually type the dollar sign, for example, when you change the Device Configuration Block.

.RS232 is the name of this serial driver. RS-232-C, on the other hand, is the name of a communications standard developed by the Electronic Industries Association. This standard defines the characteristics of signals sent between communications equipment (modems or their equivalent) and the computers, printers, terminals, stock tickers or what have you that are connected via such modems.

You can purchase a copy of this standard by writing to: EIA Engineering Department, Electronic Industries Association, 2001 Eye Street NW, Washington, D.C. 20006.

The next section provides a table of Device Configuration Block values for connecting several commonly used printers and terminals, as well as for connecting another Apple III or an Apple II. This table summarizes the information you need to connect the devices listed. Subsequent sections explain how the driver works, how to control the device from a computer program, how to select and set up a protocol for serial devices in general, and how to use advanced techniques to check the status of the device and change its control parameters.

Setting Up for Commonly Used Devices

The twelve Device Configuration Block parameters used by the .RS232 driver are explained fully in a later section of this chapter. However, Table 6-1 is a ready-made list of DCB values for several commonly used devices. For these devices, all you need to do is boot the Utilities diskette, select the Edit Driver Parameters option, and enter the Device Configuration Block parameters, following the instructions given in Chapter 2. Then make sure the switches and thumbwheels, etc., on the connected device are set for the characteristics given in parentheses.

All setups shown are typical local connections, and use a modem eliminator. Device Configuration Block parameters that have no effect under a given protocol are shown as "xx": these parameters can be left set to any value.

Device and characteristics (baud,bits,parity,protocol)	Device Configuration Block byte: 00	01	02	03	04	05	06	07	08	09	0A	0B
Default values (300,7,odd,no protocol)	06	22	00	00	00	00	13	11	DF	84	50	00
Another Apple III (9600,7,odd,hdwr handshake)	0E	22	00	00	00	00	xx	xx	xx	xx	xx	80
Apple II (300,7,SPACE,no protocol)	06	2E	00	00	00	00	xx	xx	xx	xx	xx	00
DEC LA120 terminal (1200,7,SPACE,XON/XOFF)	08	2E	00	00	00	80	13	11	DF	84	xx	00
DEC VT100 terminal (9600,7,SPACE,XON/XOFF)	0E	2E	00	00	00	80	13	11	DF	84	xx	00
SOROC IQ120 terminal (9600,7,MARK,no protocol)	0E	2A	00	00	00	00	xx	xx	xx	xx	xx	00
Qume Sprint 5 printer (1200,7,odd,hdwr handshake)	08	22	00	00	00	00	xx	xx	DF	84	xx	80
Qume Sprint 5 printer (1200,7,odd,ETX/ACK)	08	22	00	00	00	40	03	06	xx	xx	6E	00
HP 7225 Plotter (2400,7,SPACE,hdwr hdshk)	0A	2E	00	00	00	00	xx	xx	DF	84	xx	80

Figure 6-1. DCB Values for Commonly Used Devices

Note that there are two sets of DCB values for the Qume Sprint 5 printer: one for hardware handshaking, the other for an ETX/ACK software protocol. Many devices, in fact, can be run using more than one type of protocol. Note also that if you want to you can install the .RS232 driver in the SOS.DRIVER file, leave out the .PRINTER driver, and configure the .RS232 driver for a Qume printer or other devices as needed.

Using the .RS232 Driver

This section discusses how to open and close the .RS232 driver's file, and how the driver performs input and output in response to BASIC and Pascal program instructions. For further details regarding BASIC, Pascal, or whatever other programming language you will be using, consult the appropriate tutorial or reference manual. This manual gives representative examples only.

Opening the Driver's File

Before you can transfer information between the Apple III and a device, you must first open the device's driver file from the program. For example, issue an OPEN# statement in a BASIC program:

```
10 OPEN#1, ".RS232"
```

In BASIC, an OPEN# ordinarily turns IS_NEWLINE ON and places a carriage return character ($0D) in NEWLINE.

To open the device's driver file in Pascal, define the file as a variable, and then RESET the variable's identifier as '.RS232' :

```
VAR IOFILE:INTERACTIVE;
  (other definitions, etc.)
BEGIN
  RESET (IOFILE,'.RS232');
  (further program instructions)
END;
```

When you open the .RS232 driver, it automatically prepares for a read operation. It resets the Device Configuration Block parameters to the

values you set using the System Configuration Program, clears both the input and the output buffer, and prepares the RS232 port to respond to news of incoming data (called a "receive interrupt").

In the following three sections, there are references to control codes and status codes. These are explained in a later part of this chapter in the section titled Advanced Techniques.

Read Operations

Once the .RS232 driver has been opened, the input buffer (an area set aside for temporary storage of incoming data) is active: any characters received—up to 255 of them—are kept in this input buffer until they are retrieved by a program instruction such as a BASIC INPUT# or GET# statement, or a Pascal READLN or READ procedure:

```
20 INPUT#1; INSTRING$          (in BASIC)

READLN(IOFILE,LINEBUF);        (in Pascal)
```

The .RS232 driver makes use of four items of information to control a read operation:

- the number of characters (bytes) to read: this varies with the application program instruction that caused the read operation to take place
- a software "switch" called IS_NEWLINE, set using control code 2: when this byte is set to $80 it is ON; when set to $00 it is OFF
- a character stored in a byte called NEWLINE, also set using control code 2: if IS_NEWLINE is ON, the driver stops reading as soon as it reads a character that matches this NEWLINE character; if the IS_NEWLINE switch is OFF, NEWLINE is ignored
- a software "switch" called IMREAD, set using control code 1: when this byte is set to $80, it is ON; when set to $00 it is OFF

A read operation proceeds like this:

- The driver transfers characters from the input buffer of the driver into the user program's buffer until the requested number of characters has been transferred; then the driver returns control to the user program.
- If IS_NEWLINE has been turned ON, the driver checks each incoming character, looking for a match with the NEWLINE character. If it finds one, the read operation ends with that character, even if the requested character count has not yet been reached.
- If IMREAD has been turned ON, the driver reads what is in the input buffer at the moment, and does not delay the user program if what is there does not fulfill the character count or include a match with NEWLINE. The driver then returns control to the user program immediately.

After completion of the read operation, the user program can check whether any errors occurred (status code 1), and how many characters were read (status code 3).

Write Operations

When a write request is made as a result of a BASIC PRINT# or OUTPUT# statement, or a Pascal WRITE or WRITELN procedure, the driver transfers characters from the user program buffer to its own output buffer and then returns control to the user program, freeing it to perform other tasks. Meanwhile, the driver transmits the characters from its output buffer to the device. Here are examples in BASIC and Pascal:

```
30 PRINT#1; OUTSTRING$                    (in BASIC)
WRITELN (IOFILE,LINEBUF);                 (in Pascal)
```

If there is not enough room left in the output buffer to hold all the characters to be transferred, the driver, and in turn the user program, waits until room becomes available. To avoid waiting for an indefinite period, the user program can check for room before issuing the write

statement by incorporating a timed loop that keeps checking for space in the output buffer (by issuing status code 3 to find out how many bytes are in the output buffer). If there is no room, the program can at some point choose to abandon the attempted write operation.

Closing the Driver's File

At some point the user program must close all drivers it has opened. In both BASIC and Pascal, this is done with a CLOSE statement:

```
40 CLOSE#1                (in BASIC)
CLOSE (IOFILE);           (in Pascal)
```

At the moment that it is closed, the driver may be awaiting completion of character transmission from the output buffer. The user program in turn will also wait until all the characters have been transmitted. Depending on the communications protocol used (see next section), the user program may wait indefinitely before resuming execution (for example, if the printer is out of order).

One way to avoid this condition is to issue a status code 3, and check to see if there are still characters in the output buffer. If there are, you can start a timed wait and periodically recheck status. If necessary, issue a control code 0 (Reset) to clear the buffer, and then close the driver.

Communication Protocols

The .RS232 driver can operate in one of four modes:

- using no protocol or handshake (the default or "normal" mode)
- using the XON/XOFF software protocol
- using the ENQ/ACK (or ETX/ACK) software protocol
- using a hardware handshake protocol

To decide on the protocol to use, read the following four sections, and consult the manuals describing the device you are connecting.

You can use a modem in any mode except hardware handshake mode.

For each type of protocol explained below, you may have to change one or more configuration block parameters. These parameters are explained in the next major section of this chapter, Changing the Configuration Block.

No Protocol

The RS232 configuration block is normally set up for simple serial input and output, using no protocol. This is the preferred mode, because of its simplicity, unless some form of control over the flow of characters is required. If the quality of the communication lines is very good, a software protocol (XON/XOFF or ETX/ACK) may be used. If the lines are poor and an expected control character (for example, XON) comes through garbled, the driver will "hang" and you may have to reboot the system to correct the problem. In such a case, you should use a hardware handshake protocol to achieve reliable control.

The XON/XOFF Protocol

This protocol is used with many popular terminals, as well as with several of the larger data communication networks. Under this protocol, the Apple III can transfer continuous strings of characters pausing only if its own or the other device's input buffer is nearly full. It is a full two-way protocol.

For the XON/XOFF protocol, modify the configuration block parameters, if necessary, as follows:

- set communication protocol to $80
- set control character 1 to $13 (DC3; "XOFF")
- set control character 2 to $11 (DC1; "XON")

- set maximum buffer level to $DF (223 characters)
- set minimum buffer level to $84 (132 characters)

Input Buffer Control

As characters are received, the .RS232 driver loads them into its own input buffer for retrieval by the user program. If character retrieval falls behind so much that the input buffer is almost full (that is, its character count reaches the maximum buffer level), the driver transmits the XOFF character to the sending device, signalling that it should suspend character transmission. When the user program has retrieved enough characters to bring the input buffer count down to the minimum buffer level, the driver transmits the XON character to the sending device, signalling that it may resume transmission.

Output Buffer Control

Under this protocol, the driver sends data from the output buffer to the receiving device at the selected baud rate. If the driver receives an XOFF character, it suspends data transmission (usually within one or two character times) until it receives an XON character from the receiving device. At this point, if there are any characters still in the output buffer, the driver resumes transmission.

The maximum and minimum buffer levels shown for the XON/XOFF protocol are those that three out of four doctors recommend.

The ENQ/ACK (or ETX/ACK) Protocol

This protocol supports a variety of widely used terminals. Under this protocol, the Apple III transfers characters in blocks of a fixed length, rather than in a continuous stream. The Apple III configuration block must specify a block length no greater than the buffer size of the device to be attached. (Consult the documentation for the device you are using.)

The driver implements this protocol for output only. If you want the Apple III to emulate an ENQ/ACK (or ETX/ACK) device to receive data, your program must include a routine that scans input characters

looking for ENQ (or ETX), and that then sends an ACK character as soon as there is sufficient room in the input buffer for a block of data.

For this protocol, you need to set these configuration block parameters:

- set communication protocol to $40
- set control character 1 to $05 (ENQ) or $03 (ETX)
- set control character 2 to $06 (ACK)
- set the correct data block length (default is $50; that is, 80 characters)

The sending device (normally the Apple III) first transmits an ENQ or ETX character to the receiving device. The receiving device responds by transmitting an ACK character only when it is ready to receive a complete block of data. The sender then transmits a block. This three-step protocol continues until there is no more data to send. (The last block can of course be shorter than the specified block length.)

The Hardware Handshake Protocol

In Hardware Handshake mode, the device driver monitors the RS-232-C handshake lines for control signals, rather than examining the incoming data lines for control characters. The device driver monitors the Data Set Ready (DSR) and the Data Carrier Detect (DCD) signals on the Apple III RS-232-C connector. If either signal goes false, the driver suspends transmission of characters until both signals again go true. (DSR and DCD are ignored in the other modes.) This arrangement works very well if you want to connect one Apple III to another local Apple III.

When the driver is initialized but not yet open, the signal Request To Send (RTS) is false. When the driver is open, RTS is normally true; however, it goes false if the character count in the input buffer exceeds the maximum buffer level (the ninth parameter: see next section) on input, and it then stays false until the character count again falls below the minimum buffer level (the tenth parameter).

Because of the characteristics of the Apple III RS-232-C port, output is suspended when RTS is false.

When you close the driver, RTS goes false.

The RTS signal appears as DCD to another Apple III connected through a modem eliminator. Thus "DCD false" signals to the transmitting Apple III to suspend transmission until both DCD and DSR are again true.

When the driver is open, Data Terminal Ready (DTR) is true. When the driver is closed, or initialized (after booting) but not yet open, DTR is false.

For this mode, modify the configuration block parameters as follows:

- set communication protocol to $00
- set hardware handshake to $80

Using a Modem

The term "modem" in this discussion means a Bell 103 or 212 modem, or any other full-duplex modem that handles the RS-232-C signals in a similar manner.

You can use a modem in any mode except hardware handshake mode. The state and interpretation of the signals are the same with a modem as in hardware handshake mode, with the following exceptions.

The signal Request to Send (RTS) is always true when the driver is open. It is false when the driver is closed.

If you are using a modem connected to a switched telephone system, it is a good idea for your program to check bit 5 of the fourteenth parameter (interface status: see next section) whenever you issue status code 1 (described under Advanced

Techniques). If bit 5 = Ø (OFF), DCD is present, meaning the call is OK, and so your program can proceed to send data or look at the input buffer. To know when the call is over, look for bit 5 = 1 (ON); bit 5 ON indicates that the carrier signal is no longer present. This normally means that the connection has been broken or that the other party has hung up the phone. Your program should take appropriate action.

The interface hardware cannot transmit data while Clear to Send (CTS) is false. If this is a potential problem, always check the input buffer level (status code 3; seventh and eighth bytes) before attempting to transmit data.

Using a Modem Eliminator

If you are not connecting a modem to the RS-232-C port, but are instead connecting a device directly to the Apple, you must connect a Modem Eliminator (the one furnished with the Apple III or an equivalent) to the RS-232-C port on the Apple, and then connect the device's cable to the Modem Eliminator.

If you connect the Apple III directly to another device, use only ONE Modem Eliminator. For example, if you connect two Apple IIIs locally, attach a Modem Eliminator to one of them, and a cable from the Modem Eliminator directly to the other Apple III. If you use two Modem Eliminators, they cancel each other out, and you end up with both devices trying to send on the same wire, receive on the same wire, and so on.

Changing the Configuration Block

The RS232 driver's configuration block has twelve parameters. The definitions and normal values of the first five parameters are identical to those of the five parameters in the printer driver's configuration block except that the default value of the first parameter, the baud rate, is $Ø6 (3ØØ baud) instead of $Ø8 (12ØØ baud). The general procedure for changing the configuration block is the same as well.

The number to the left of each parameter in Figure 6-2 is in decimal and is used in this discussion. The number to the right is the hexadecimal number used by the System Configuration Program (SCP).

dec		SCP #
1	Baud Rate	ØØ
2	Data Format	Ø1
3	Carriage Return Delay	Ø2
4	Line Feed Delay	Ø3
5	Form Feed Delay	Ø4
6	Communication Protocol	Ø5
7	Control Character 1	Ø6
8	Control Character 2	Ø7
9	Maximum Buffer Level	Ø8
1Ø	Minimum Buffer Level	Ø9
11	Data Block Length	ØA
12	Hardware Handshake	ØB

Figure 6-2. The .RS232 Device Configuration Block

The first parameter is the baud rate. This parameter controls the speed at which the Apple sends and receives bits of information. There are nine possible speeds, measured in baud.

Value	Speed	
Ø3	11Ø	baud (Teletypewriter speed)
Ø4	134.5	baud
Ø6	3ØØ	baud (A common telecommunications speed)
Ø7	6ØØ	baud
Ø8	12ØØ	baud (A common printer speed)
Ø9	18ØØ	baud
ØA	24ØØ	baud
ØC	48ØØ	baud
ØE	96ØØ	baud

The default value is $Ø6, 3ØØ baud. To change the speed, use the four arrow keys to position the indicator (a pair of square brackets) on the first value in the configuration block. Type the two characters of the new speed and press RETURN.

The second value is the data format. This determines the arrangement of bits in the data stream that the computer sends to the device. There are nine standard formats:

Value	Format
22	7 bits, odd parity (odd number of 1's)
26	7 bits, even parity (even number of 1's)
2A	7 bits, MARK parity (parity bit always 1)
2E	7 bits, SPACE parity (parity bit always Ø)
ØØ	8 bits, no parity
42	6 bits, odd parity (odd number of 1's)
46	6 bits, even parity (even number of 1's)
4A	6 bits, MARK parity (parity bit always 1)
4E	6 bits, SPACE parity (parity bit always Ø)

If you set the first parameter to 11Ø baud ($Ø3), the driver will force two stop bits regardless of the value of this parameter. At all other band rates, the driver will use one stop bit.

For ASCII input, select 7 bits plus the appropriate parity. For binary data transfer, 8 bits is normally your best bet.

Warning: the driver passes ALL characters of data, including control characters, to and from the user program's language buffer.

The default value is $22: seven data bits per character, with the high bit set to 1 or Ø as necessary to make the total number of 1's in the byte odd. The parity bit is used for error checking.

The third parameter determines the number of character-times the RS232 driver is to wait after sending a carriage return character (ASCII CR; value $0D). This allows the device enough time (if it needs it) to prepare itself to receive the next line.

The default carriage return delay value is $00 (no delay). The delay amount can range from 0 through 255, but you must specify it in hexadecimal—that is, base 16.

The fourth parameter is similar to the third: it determines the number of character-times the RS232 driver is to wait after sending the line feed character (ASCII LF; value $0A). This allows the receiving device the time it needs to advance one line. The default value is $00 (no delay).

The fifth parameter is also similar to the third: this one determines the number of character-times the RS232 driver is to wait after sending a form feed character (ASCII FF; value $0C). This allows the receiving device time (if it needs it) to advance to the top of the next printing page. The default value is $00 (no delay).

The sixth parameter specifies the communications protocol the driver should use: no protocol ($00), the XON/XOFF protocol ($80), or the ENQ/ACK protocol ($40). This parameter must be $00 if the driver is to operate in hardware handshake mode (see parameter twelve below).

The default value of this parameter is $00, no protocol.

The seventh parameter contains control character 1: for the XON/XOFF protocol, set this parameter to $13 (XOFF); for the ENQ/ACK protocol, set it to $05 (ENQ); for the ETX/ACK protocol, set it to $03 (ETX).

The default value of this parameter is $13, XOFF.

The eighth parameter contains control character 2: for the XON/XOFF protocol, set this parameter to $11 (XON); for the ENQ/ACK or ETX/ACK protocol, set it to $06 (ACK).

The default value of this parameter is $11, XON.

The ninth parameter specifies the maximum buffer level for the XON/XOFF protocol: when the input buffer has this number of characters in it, the driver will transmit the XOFF character to the sending device.

The default maximum buffer level is $DF (223 characters). Since the input buffer can hold 255 characters, this allows room for 32 additional characters, and thus enough time for the sender to respond to the XOFF character before overflowing the driver input buffer, even at fast transfer rates.

The tenth parameter specifies the minimum buffer level for the XON/XOFF protocol: when the character count in the input buffer falls below this level after transmission of an XOFF, the driver transmits an XON to resume transmission of characters to the Apple.

The default minimum buffer level is $84 (132 characters).

The eleventh parameter specifies the data block length for the ENQ/ACK or ETX/ACK protocol. You can set it to any value from $Ø1 through $FF (255 characters per block), but this value must not exceed the buffer size of the receiving device.

The default data block length is $5Ø (8Ø characters per block).

The twelfth parameter indicates to the driver whether to operate in hardware handshake mode ($8Ø) or not ($ØØ). Set the sixth parameter to $ØØ if you set this one to $8Ø.

The default value of this parameter is $ØØ.

Advanced Techniques

System calls are the normal method that assembly-language programs (including interpreters and language systems) use for communicating with files and devices on the Apple III.

You can control the operation of the device connected via the RS232 driver by issuing your own system calls. The particular system calls

that you use to control device drivers are D_STATUS and D_CONTROL.

Interpreters (such as BASIC) and language systems (such as Pascal) provide more convenient ways of communicating with files and devices, so that user programs don't ordinarily need to resort to system calls. However, some language systems on the Apple III provide ways for you to make SOS calls directly to the system. For example, Apple III Pascal includes a procedure called UNITSTATUS that is used for both D_STATUS and D_CONTROL system calls. The use of this procedure is described in Appendix H.

These system calls require three parameters, like this:

D_STATUS (device number, status code, status list)

D_CONTROL (device number, control code, control list)

where device number specifies the device you want to get status information from or send control information to; status code and control code specify the particular information to get or to send; and status list and control list specify the location in the program's memory space where the information is or will be put.

Status Requests

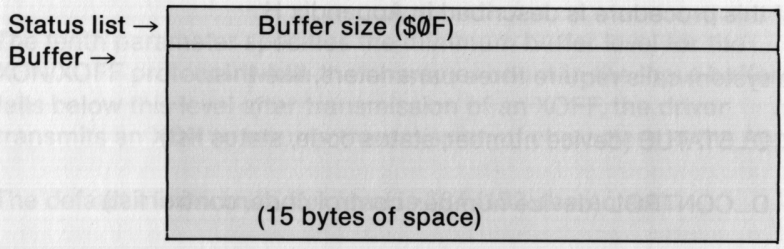
The following list gives the status code and the contents of the status list for each RS232 driver status request.

Status code: 0 (No Operation)
Status list: (nil)

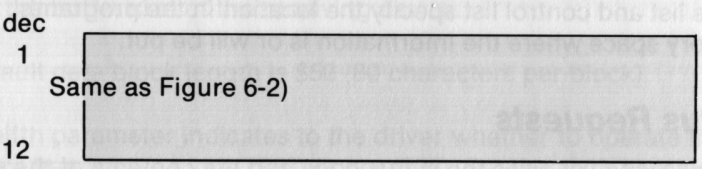
This status request does nothing.

Status code: 1 (Retrieve Control Parameters)
Status list: Buffer size, Buffer

Copies the fifteen RS232 control parameters into the buffer portion of status list. The first byte of status list (buffer size) must be greater than or equal to 15 ($ØF), the number of control parameters. At the end of the status call, this first byte will contain the actual number of bytes used (15).



The first twelve control parameters copied into the buffer are the parameters of the RS232 driver's configuration block:



The thirteenth parameter, IMREAD, controls whether a read request is simply to retrieve all characters currently in the buffer and then continue (IMREAD=$8Ø) or wait until the request is satisfied (IMREAD=$ØØ)—that is, until it has either retrieved the requested number of characters, or detected the NEWLINE character (if IS_NEWLINE=$8Ø) in the incoming stream of characters.

13 Immediate Read

The fourteenth parameter is a byte (eight bits) indicating the status of the interface at the time of the most recent interrupt:

14 Status At Last Interrupt

7	6	5	4	3	2	1	0
IRQ	DSR	DCD	Xmt Reg	Rcv Reg	Overrun	Frm Err	Par Err

Bit	"1" Means	"0" Means
0	Parity error occurred	No parity error
1	Framing error occurred	No framing error
2	Overrun occurred	No overrun
3	Receiver data register full	Register not full
4	Transmitter data register empty	Register not empty
5	Data Carrier Detect (DCD) false	DCD true
6	Data Set Ready (DSR) false	DSR true
7	Interrupt request occurred	No interrupt occurred

The fifteenth parameter is a "latched status" register; that is, it contains the accumulation of selected status bits since the most recent status code 1 or control code 0 request cleared this register. Bits 0, 1, 2, 5 and 6 have the same meanings as their corresponding bits in the status byte (the fourteenth parameter). Bit 7 equals 1 if one or more input characters has been lost due to an input buffer overflow; it equals 0 of no characters were lost. Bits 3 and 4 have no meaning.

15	Latched Status

7	6	5	4	3	2	1	0
O'flow	DSR	DCD			Overrun	Frm Err	Par Err

Once you have retrieved the control parameters, you can examine or even change them. You can restore their original values, or change the values in memory and then put them in the Device Control Block, by issuing a Set Control Parameters request (control code 1).

Status code: 2 (Retrieve Newline Information)
Status list: Buffer

Places two bytes—the IS_NEWLINE flag and the actual NEWLINE character—in the first two bytes at the location you specify in the request. See the section of this chapter on Read Operations for an explanation of these two bytes.

Status code: 3 (Retrieve Driver Buffer Information)
Status list: Buffer

Loads eight bytes of buffer information into the location you specify in the request.

1	Output buffer size (low-order)
2	Output buffer size (high-order)
3	Number of characters in output buffer (low)
4	Number of characters in output buffer (high)
5	Input buffer size (low-order)
6	Input buffer size (high-order)
7	Number of characters in input buffer (low)
8	Number of characters in input buffer (high)

RS232 Control Requests

The following list gives the control code and the contents of the control list for each RS232 driver control request.

Control code: Ø (Reset RS232 Driver)
Control list: (nil)

Clears output and input buffers and resets the RS232 driver. The current control parameters values are used for the reset. These values are the parameters set using the System Configuration Program (default), or those set by the most recent control code 1 call. Any characters not yet transmitted from the output buffer or retrieved from the input buffer are lost.

Control code: 1 (Set Control Parameters)
Control list: Buffer size, Buffer

Loads the first thirteen of the fifteen device control parameters into the driver from the buffer indicated. The control parameters are in the same format as described under status code 1. The interface status byte (the fourteenth parameter) and "latched status" register (the fifteenth parameter) are not loaded; instead, a reset (control code Ø) is automatically performed prior to completing this request.

Buffer size is the byte count returned by a status code 1 request. This first byte must equal $ØF or an error will result.

The buffer should contain the information put there by a prior Retrieve Control Parameters Request; it is not advisable to enter your own values into this table unless you need to change them (for example, change the baud rate).

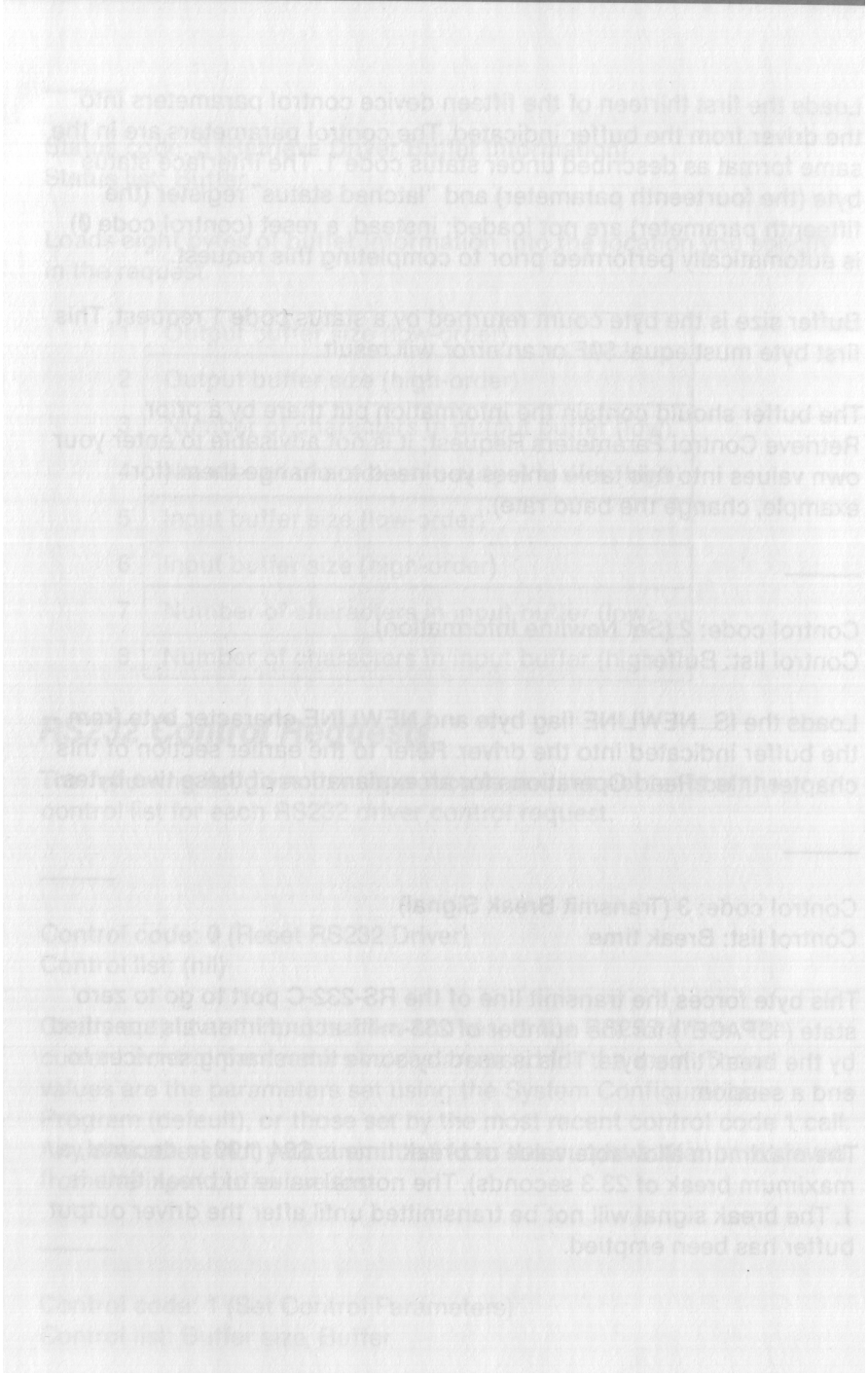
Control code: 2 (Set Newline Information)
Control list: Buffer

Loads the IS_NEWLINE flag byte and NEWLINE character byte from the buffer indicated into the driver. Refer to the earlier section of this chapter titled Read Operations for an explanation of these two bytes.

Control code: 3 (Transmit Break Signal)
Control list: Break time

This byte forces the transmit line of the RS-232-C port to go to zero state ("SPACE") for the number of 233-millisecond intervals specified by the break time byte. This is used by some timesharing services to end a session.

The maximum allowable value of break time is $64 (1ØØ in decimal, a maximum break of 23.3 seconds). The normal value of break time is 1. The break signal will not be transmitted until after the driver output buffer has been emptied.



7

The Audio Driver

The standard device driver .AUDIO enables you to produce tones from the Apple III's built-in speaker. You can control three aspects of the tones: their volume, pitch, and duration.

The audio driver is an output device only; you cannot accept input from the .AUDIO file.

Data is sent to the driver in the form of characters. The audio driver examines the characters for a special character that signifies the beginning of a tone specification. It then takes the five characters following the special character and produces a tone determined by the values of those characters. After the tone is completed, the driver examines the remaining characters for another special character. If there are no more characters to process, it returns control to the program that requested audio output.

The audio driver does not respond to any other characters, nor can it process normal text. There are no control or status requests, and there are no values in the audio driver's configuration block that can be changed by the System Configuration program.

Tone Parameters

To request the audio driver to generate a tone, you send it four parameters:

Parameter	Definition
Mode	Type of request
Volume	Volume of the tone
Count	Pitch of the tone
Time	Duration of the tone

The *mode* parameter indicates to the audio driver that this is the beginning of a request, and specifies what kind of request you are making. The audio driver supports only one mode at this time: simple tone generation, mode 128. All requests to the audio driver should be made with 128 as the value of the mode parameter.

The value of the *volume* parameter specifies how loud the tone should be. There are 64 volume settings, ranging from Ø (silent) to 63 (fairly loud).

The value of the *count* parameter determines the pitch of the tone. The lower the count value is, the higher the pitch of the tone is, as determined by this formula:

$$\text{freq} = \frac{50900}{\text{count}}$$

The count value can range from 16383 to 100: this gives a range of frequencies from about 31 Hz to over 5000 Hz. In musical terms, this is over seven octaves, from the C three octaves below middle C to the E flat four octaves above middle C.

The value of the *time* parameter determines the duration of the tone, in sixtieths of a second. The time value can range from Ø to 300; a value of Ø produces silence, while a value of 300 produces a tone lasting about five seconds, as determined by the formula:

$$\text{duration} = \frac{\text{time}}{60}$$

Producing Tones

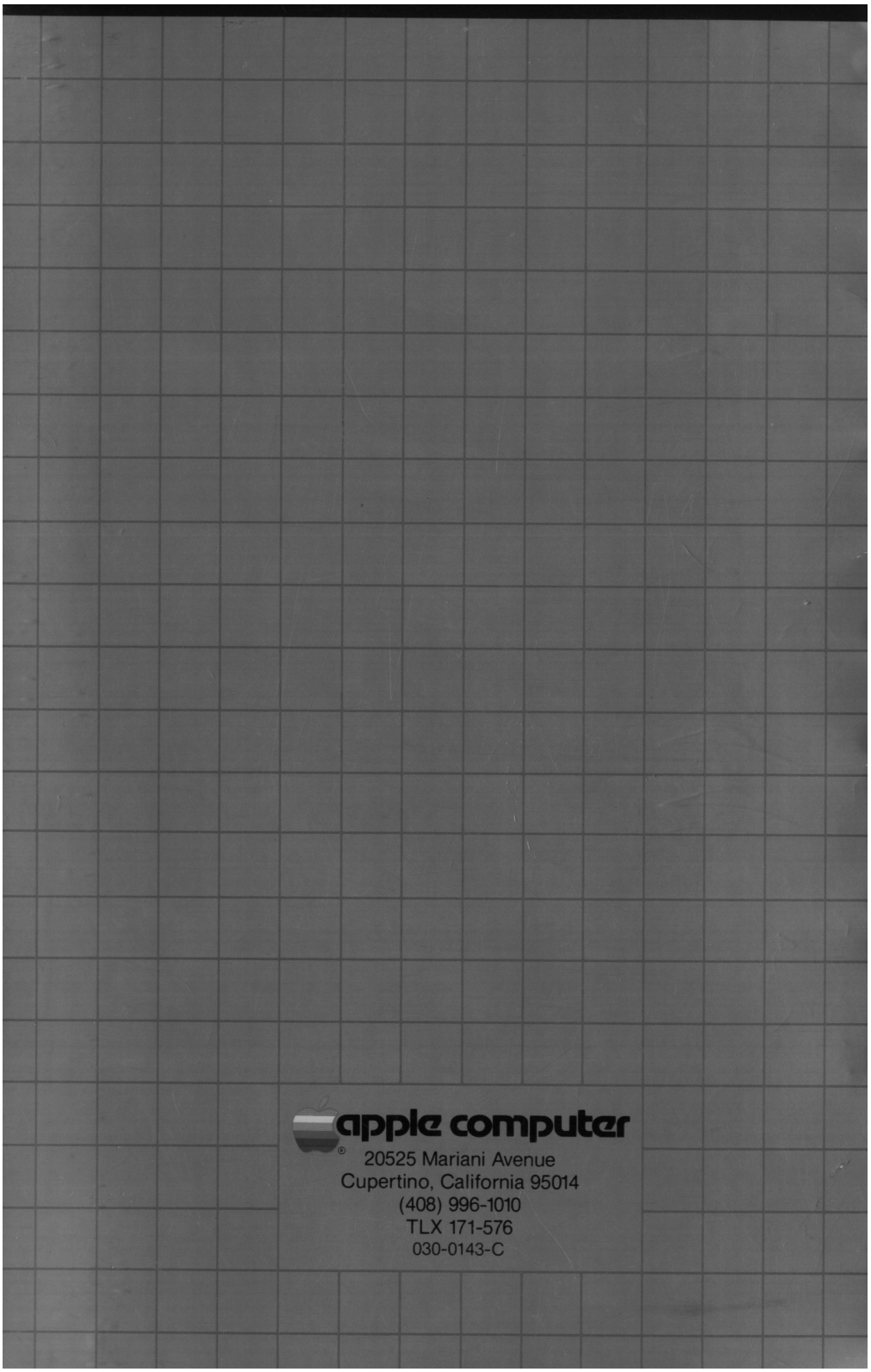
The mode, volume, count, and time of a tone are sent to the audio driver in the form of a six-character string. The mode and volume are each represented by one character; the count and time are each represented by two characters in the string. The values of the parameters are determined by the ASCII values of the characters. This is the arrangement of the characters in the string:

Character:	1	2	3	4	5	6
Max. Value:	128	63	255	63	44	1
Meaning:	Mode	Vol.	Count		Time	

The count and time values are two-byte integers, with the low-order byte first. The maximum values shown for these parameters are the two-byte values: the low-order byte of the time value can range from Ø to 255, but the maximum time value is 3ØØ. This is obtained by adding a low-order byte of 44 to the value of a high-order byte of 1, that is, $1ØØ or 256.

A BASIC program using the variables MODE%, VOL%, COUNT%, and TIME% would form a string in the above format with the statements

```
OPEN#1 AS OUTPUT, ".AUDIO"
PRINT#1;CHR$(MODE%);CHR$(VOL%);
PRINT#1;CHR$(COUNT%-256*INT
(COUNT%/256));CHR$(INT(COUNT%/256));
PRINT#1;CHR$(TIME%-256*INT
(TIME%/256));CHR$(INT(TIME%/256));
```


apple computer
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576
030-0143-C