P R O M A L

(**PRO**grammer's **M**icro **A**pplication **L**anguage)

USER'S GUIDE

A GUIDE TO USING THE PROMAL

-- EXECUTIVE --
-- EDITOR --
-- COMPILER --

For Apple II and Commodore 64 Computers

SYSTEMS MANAGEMENT ASSOCIATES, INC.
3325 Executive Drive
Raleigh, North Carolina 27609
(919) 878-3600

Rev. C - September 1986

PROMAL USER'S GUIDE

INTRODUCTION

Welcome to the exciting world of programming in PROMAL. PROMAL provides a **complete programming environment** to let you get the most out of your computer. Unlike other programming languages which have historically come from big computers and have been "shoehorned" into personal computers, PROMAL was designed from the ground up for use on small machines. Because of this, it provides the tools you need to quickly and easily write programs which run "lightning-fast".

Not only do PROMAL programs often run 20 to 100 times (or more) faster than BASIC, but non-trivial programs are usually easier to program and maintain in PROMAL than BASIC.

Your PROMAL programming system includes all of the following:

* An operating system EXECUTIVE for interactive control
* A powerful full-screen program EDITOR for preparing programs
* A fast, one-pass COMPILER for the PROMAL language
* A standard LIBRARY of over 50 versatile subroutines, ready to use
* Ready-to-run demonstration programs for you to run, study, and modify
* A complete manual with examples to guide you

Let's take a quick look at what each of these tools does for you.

The PROMAL operating system **EXECUTIVE** is your control center. You type in commands to run programs, activate the PROMAL editor or compiler, and perform other operations. A number of built-in commands are provided for manipulating files, displaying and changing memory, etc. In addition, you can add your own commands. The PROMAL EXECUTIVE lets you have several programs in memory at once, and you can run any of them instantly by merely typing the name of the program.

The PROMAL full screen **EDITOR** makes it easy to create or change programs or text information. The editor is like a good word processor, except that it is designed specifically for writing PROMAL programs. To make changes, you simply move the cursor around on the screen and insert or delete text as desired. The editor can scroll forwards or backwards to rapidly display any part of your program. Powerful search and replace commands make it easy to make corrections. You can "cut and paste" blocks of text, too.

The most important part of the system is the PROMAL language and compiler. The **COMPILER** takes the program you created with the EDITOR and converts it to a form which runs with nearly the speed of assembled machine language programs. It is this compilation process which is mainly responsible for PROMAL's great speed. Compiled PROMAL programs also occupy less memory than BASIC or other languages. You can save your compiled program on disk, and it can be run at any time later by just typing its name.

The **LIBRARY** of over 50 subroutines is already built into the PROMAL system. You can call these routines from your program to perform a wide variety of tasks. Having all these subroutines immediately available greatly simplifies the job of programming.

There's nothing like some good examples to speed the learning process, so several complete, useful **demonstration programs** are included on the PROMAL disk.

ABOUT YOUR PROMAL MANUALS

If you have not already done so, you should read your **MEET PROMAL!** manual, which will provide you with a "hands-on" guided tour of the PROMAL system as a whole. This short manual introduces many of the novel concepts of the PROMAL system in a "get-acquainted" style.

This manual, the PROMAL USER'S GUIDE, tells you how to use the various components of the PROMAL system. After this introduction, it is divided into three major sections, covering:

1). Operation of the EXECUTIVE (including all EXECUTIVE commands);
2). Operation of the EDITOR;
3). Operation of the COMPILER.

The PROMAL LANGUAGE MANUAL, describes the PROMAL language in detail. You will probably want to read it after you have gained some proficiency with the EDITOR and EXECUTIVE by reading the USER'S GUIDE.

The PROMAL LIBRARY MANUAL provides a detailed reference for the the subroutines in the LIBRARY, arranged alphabetically. You will want to skim this material after reading the LANGUAGE MANUAL, and refer back to it for details as the need arises.

A set of Appendices serves all of these manuals, giving supplemementary information. An index is provided to all manuals and the Appenidces.

If you purchased the Developer's Package, you will have an additional manual, the DEVELOPER'S GUIDE, which covers topics relevent to making stand-alone PROMAL programs which can be run on systems without PROMAL.

STARTING THE SYSTEM

The **MEET PROMAL!** manual tells you how to "boot up your system" using a copy of the Demo disk. You should always use a "working disk" that has a copy of the PROMAL software on it, and leave the original disk in a safe place.

Once you have your system booted up, PROMAL will sign on and the EXECUTIVE will now display the default meanings of the function keys (note: see the JOB command description below for a way to defeat the display of the default function keys during boot-up). This display will look similar to this:

Commodore	Apple II
F1 = EDIT	F1 = EDIT
F2 = DUMP	F2 = PREFIX *
F3 = COMPILE	F3 = COMPILE
F4 = GET	F4 = GET
F5 = FILES	F5 = FILES
F6 = MAP	F6 = EXTDIR *
F7 = HELP	F7 = HELP
F8 = COPY	F8 = COPY

Note: F4 means hold either Apple key and press 4.

If you have read **MEET PROMAL!**, you already know that pressing one of the function keys is equivalent to typing in the command name it stands for. For example, pressing F1 will cause the word "EDIT" to appear on the screen, just as if you typed it in. The "-->" is the PROMAL EXECUTIVE prompt, followed by a blinking cursor. This indicates that the EXECUTIVE is waiting for you to type in a command.

You are now ready to begin using PROMAL.

EXECUTIVE COMMANDS AND COMMAND EDITING

To tell the EXECUTIVE something, you can use any of the function keys or type in a command that the EXECUTIVE knows. The "built-in" commands are described in detail in the next section. All commands must be terminated by the RETURN key. Up until the time you press the RETURN key, you can use any of the line-editing keys described in **Table 1** to make corrections.

Once you press RETURN, the EXECUTIVE will attempt to execute whatever command you have typed. There are three "levels" of commands, which the EXECUTIVE searches in this order:

1). Built-in commands
2). User-defined commands in memory
3). User-defined commands on disk

If the EXECUTIVE can not find the command in any of these places, it will display:

```
*** ERROR: PROGRAM OR OVERLAY NOT FOUND: xxxx
--> _
```

User-defined commands are simply compiled PROMAL programs. Several of these command files are on the Demo disk. They are easily recognized because their names end in ".C", indicating a command file. You can create your own commands by writing a PROMAL program and compiling it. To run a PROMAL program, you don't have to LOAD it and RUN it like you do with BASIC; you simply type its name. The EDITOR and COMPILER sections of this manual and the LANGUAGE MANUAL contain all the information you need to create your own commands.

Commands may be typed in either upper or lower case letters. PROMAL operates using upper-and-lower case, like a normal typewriter. If you prefer all upper case letters, you can press CTRL A, which enables alpha-lock.

TABLE 1

PROMAL LINE-EDITING KEYS

Commodore Key	Apple Key	Description
RETURN	**RETURN**	End of line. The completed line is entered into the PROMAL system. May be typed from any cursor position in the line. Maximum line size is 80 characters for the EXECUTIVE.
DEL	**DELETE**	Replace the character left of the cursor with a blank and back up the cursor one position
INST	**CTRL E**	Enable "insert mode". Any characters subsequently typed will be inserted before the character the cursor is on, pushing any existing text to the right. Exit insert mode by pressing RETURN or other line-editing keys.
CTRL <--	**CTRL D**	Delete character with pullback. Deletes the character under the cursor and pulls any remaining text to the left to fill in the gap.
==>	**-->**	Cursor right. Moves the cursor to the right, without altering the character under the cursor. Stops at the end of the line. Repeats automatically after a brief pause if held down.
<==	**<--**	Cursor left. Moves the cursor to the left, without altering the character under the cursor. Stops at the first character entered. Repeats automatically after a brief pause if held down.
CTRL X	**CTRL X**	Cancel the entire line. Erases all characters typed on the line and repositions the cursor to the first character position.
CTRL K	**CTRL \**	Clear to end of line. Erases all characters from the cursor to the end of line.
CTRL Y	**CTRL L**	Jump to last character of line. Moves the cursor to the column after the last character on the line, without affecting the line content.

NOTE: The CTRL key is used like a shift key. CTRL X means you hold down the CTRL key and press X at the same time. N/A means not available.

TABLE 1 (continued)

Commodore Key	Apple Key	Description
CTRL [	**CTRL F**	Jump to first character of line. Moves the cursor to the first character position, without affecting the line content.
CTRL A	**CTRL A**	Toggle Alpha-Lock mode. When pressed the first time, causes all subsequent alphabetic characters to be entered and displayed as upper case when typed. Does not affect other characters displayed on the screen or already typed. Pressing CTRL A again returns to normal upper and lower case alpha mode.
F1 through F8	**Apple 1 through Apple 8**	Function key. Erases current text on command line and enters the equivalent function key definition. More can be typed after the function key definition or it can be edited further if desired.
CTRL B	**CTRL B**	Backtrack. Erases current text on the command line and enters the last line entered. More can be typed after the recalled command, or it can be edited further. Pressing CTRL B again will recall the next-to-last command entered. This can be repeated up to the limit of the backtrack buffer of 256 characters; then the display will "wrap around" to repeat the most recent command again.
CTRL STOP	**CTRL RESET**	Program abort. Unconditionally aborts the currently-running program and returns control to the PROMAL EXECUTIVE, closing any open files. Should be used only as an "emergency exit" (does not work if interrupts are disabled or if a machine-language program has executed an illegal opcode). Apple version clears screen and may cause loss of data in files open for writing.
CTRL Z	**CTRL Z**	Indicates end of file from the keyboard device if it is the first character of a line. See the TYPE command for an application.

NOTE: The CTRL key is used like a shift key. CTRL X means you hold down the CTRL key and press X at the same time. N/A means not available.

It is possible to alter almost all of the choices for editing keys (See **Appendix G**). The default keys were chosen so as not to conflict with pre-defined Commodore keys.

For the Commodore 64, CTRL-STOP will not be operational while the disk is being accessed with DYNODISK enabled, because DYNODISK disables interrupts temporarily while it is running.

TABLE 1 (continued)

SPECIAL SYSTEM-DEPENDENT KEYS

Commodore Key	Apple Key	Description
STOP	**N/A**	Stop action. This key temporarily suspends all program execution (including user-defined programs, built in commands, and machine-language programs) until the key is released. It is useful for halting a rapidly-changing display so you can read it (does not work if interrupts are disabled).
CTRL	**N/A**	Slow display. During display on the screen, slows down the scroll rate while held down. Makes it easier to read rapidly scrolling text.
N/A	**CTRL C**	Abort command. Can be used to abort an Executive command or a running program if it is displaying on the screen. This method is preferred to CTRL-RESET for aborting programs on the Apple.
N/A	**CTRL S**	Pause output. Temporarily halts display to screen until any key is pressed.

NOTE: The CTRL key is used like a shift key. CTRL X means you hold down the CTRL key and press X at the same time. N/A means not available.

BUILT-IN EXECUTIVE COMMANDS

The EXECUTIVE has a number of built-in commands which are always available, summarized in **Table 2** below. These commands are explained in detail in the following sections.

TABLE 2

BUILT-IN EXECUTIVE COMMANDS

Command	Avail*	Function
BUFFERS	**A**	Set the number of ProDOS disk buffers (# open files).
COLOR	**C**	Change the current color and/or screen background color.
COPY	**AC**	Copy a file.
CS	**AC**	Clear the screen.
DATE	**AC**	Change the current date (See note).
DELETE	**AC**	Delete a file.
DISKCMD	**C**	Send C-64 disk commands and display error channel replies.
DUMP	**AC**	Display memory in hexadecimal and ASCII characters.
DYNO	**C**	Enable/disable double speed read for 1541/1571 disk drives.
EDIT	**AC**	Enter the full screen PROMAL EDITOR.
FILES	**AC**	Display the names of files on disk.
FILL	**AC**	Fill a region of memory with a constant.
FKEY	**AC**	Redefine a function key or display present assignments.
GET	**AC**	Load a PROMAL or machine language program into memory.
GO	**AC**	Execute a machine language program in memory.
HELP	**AC**	Display a "help" menu of EXECUTIVE commands and control keys.
JOB	**AC**	Execute a list of EXECUTIVE commands stored in a file.
LOCK	**A**	Lock (write-protect) a file.
MACRO	**AC**	Define a command macro.
MAP	**AC**	Display the current memory allocation and loaded programs.
NEWDIR	**A**	Create a new directory.
NOREAL	**AC**	Discard support for REAL data (makes more memory available).
PAUSE	**AC**	Display a message and wait for RETURN key.
PREFIX	**A**	Display or change disk volume name or subdirectory.
QUIT	**AC**	Exit to BASIC (Commodore) or to specified system (Apple).
RENAME	**AC**	Change the name of a file.
SET	**AC**	Set memory locations to specified values or characters.
SIZE	**AC**	Display the size of a compiled PROMAL program.
TYPE	**AC**	Display a file of text on the screen, printer, etc.
UNLOAD	**AC**	Remove a PROMAL program from memory.
UNLOCK	**A**	Unlock (allow writing) a file.
WS	**AC**	Clear or alter the size of the Workspace (in-memory file).

***Note:** A=Apple, C=Commodore
DATE is not a built-in command on the Commodore 64, but a compiled program which is automatically unloaded after it is run.

ARGUMENTS FOR EXECUTIVE COMMANDS

Many EXECUTIVE commands have required or optional **arguments**. An argument is a series of characters separated from the command by one or more blanks. For example:

```
COPY MYFILE.T
```

has one argument, "MYFILE.T", and

```
DUMP 1000 1078
```

has two arguments. The kind of argument needed (if any) varies with the individual commands. Frequently an argument will be a file name to operate on.

In order to describe what kind of arguments a command can have, the following notation is used in this manual:

(1). A name shown in all CAPITAL letters indicates the name of the command or a word which must be typed in exactly as shown. It may be typed in either upper or lower case letters.

(2). A name shown in Upper and lower case letters is a description of something the user must type in. For example,

```
COPY Filename
```

means the argument must be a legal PROMAL filename.

(3). Anything enclosed in square brackets, "[ ]" is optional. The text will explain what the default is if the optional argument is not specified.

(4). Ellipsis (...) are used to show an arbitrary number of repetitions of the preceding item. For example:

```
SET Address Value [...]
```

means that the SET command can have an arbitrary number of Value arguments specified.

FILE NAMES

File names are frequently used as arguments for EXECUTIVE commands. File name requirements differ somewhat for various computers, because the disk formats and underlying operating systems are different. In order to promote portability between computers, PROMAL uses a default naming convention that is very similar for all computers, but may not allow access to all file names and file types that are legal on a particular computer. PROMAL normally operates on these **PROMAL files** automatically, by default. However, provision is made in the EXECUTIVE and PROMAL language to be able to operate on **any** type of file which is legal on your computer. Find the file name rules which apply to your computer below. Hereafter, any reference to a file name means a PROMAL file name unless otherwise stated.

File Names for **Commodore 64** Computers

PROMAL can operate on standard **PROMAL files**, or any kind of file available on the Commodore 64. By default, PROMAL operates on standard PROMAL files, which conform to these rules:

(1). The name must be 1 to 14 characters long, with the first character being an alphabetic character. If a single character name is chosen, it should not duplicate the names of any of the PROMAL devices given in **Table 4.** The remaining characters must be alphabetic, numeric, or the left-pointing arrow character (an ASCII underline character; this is the key above the CTRL key). The name may NOT contain blanks or other punctuation, and is **not** enclosed in quotes when used as a EXECUTIVE command argument. Names may be typed in upper or lower case, but are converted to all upper case internally.

(2). The name part can optionally be followed by a period and a single character **file extension.** The file extension must alphabetic or numeric. It indicates the "kind" of file. **If omitted, a default extension of ".C" will be assumed, which indicates a PROMAL command file (executable program).** Multiple character file extensions may be used but are not recommended.

(3). The file name may have an optional drive number prefix followed by a colon. The choices are **0:** or **1:**. If no prefix is specified, 0: is assumed. See **Appendix N** for information on multiple drive systems.

All normal PROMAL files on the Commodore 64 are stored as sequential (SEQ) type files, including executable programs. PRG and REL files are not normally used. The following are examples of legal PROMAL file names:

```
AB        x7          MyData      YOUR45.T    DOIT.C
Hello_There.C         T12345.S    Z_          RECOVER.S
0:STUFF               1:AUXDATA.D
```

The names above which do not have a file extension will be stored on disk with the file extension ".C". Thus if you type AB for a file name, the file AB.C will be the file acted upon (EXCEPTION: The EDITOR and COMPILER will assume a file extension of .S for the source files they operate on). The following names are not legal PROMAL files for the reasons noted in parentheses:

7THDATA.S	(must start with an alphabetic character)
MY Data	(can't have embedded blanks or punctuation)
THE_LAST_PROGRAM	(can't have more than 14 characters)
OLD.STUFF.D	("." can only be used to start the file extension)

EXECUTIVE commands normally use PROMAL file names. However, you may access any file name which is legal on the Commodore 64 by enclosing the name in quotes. The COPY command also requires specification of the file type for non-SEQ files, as is described later.

Appendix **M** describes PROMAL support of relative files, which should not be manipulated with EXECUTIVE commands. It is also possible to open files to access Commodore direct access files, directories, and the command/error channel. These facilities are described later.

File Names for **Apple II** Computers

PROMAL can operate on standard PROMAL files, or any kind of file available under ProDOS. By default, PROMAL operates on standard PROMAL files. A PROMAL file name is slightly more restrictive than a ProDOS file name. Legal PROMAL file names must conform to these rules:

(1). The name must be 1 to 13 characters long, with the first character being an alphabetic character. The remaining characters must be alphabetic, or numeric. The name may NOT contain blanks, periods, underlines, or other punctuation. If a single character name is chosen, it should not duplicate a PROMAL device name given in **Table 4.** Names may be upper or lower case. ProDOS converts all names to uppercase internally.

(2). The name part can optionally be followed by a period and a single character **file extension.** The file extension must alphabetic or numeric. The file extension indicates the "kind" of file. If omitted, **a default extension of ".C" will be assumed,** which indicates a PROMAL command file (executable program). Multiple character file extensions are permitted but not recommended. The total name including the extension may not exceed 15 characters.

(3). The file name may have an optional **pathname** which specifies the ProDOS volume name and/or subdirectory name. Volume names are indicated by a leading / character, are up to 15 characters long, and start with an alphabetic character. Volume names are assigned when the disk is formatted using the ProDOS Utility. Subdirectories follow the same naming rules as volumes, and can be specified when files are created. **If no prefix is specified as part of the file name, then the current prefix will be used.** The current prefix is set by the PROMAL **PREFIX** command, and is initially the prefix of the disk or directory from which PROMAL was booted. The total combined pathname and filename cannot exceed 60 characters. For floppy disks, we suggest you avoid using subdirectories. If a path name and file is specified without a leading / character, it will be appended to the present path. For example if the present prefix is /MYDISK/, and a name given is PROGS/GO.C, then the resulting path will be /MYDISK/PROGS/GO.C.

(4). In lieu of a volume name, you may use a two character drive prefix as follows:

0:	The /RAM volume (slot 3 drive 2)
1:	Floppy drive 1 (slot 6)
2:	Floppy drive 2 (slot 6)

When using the drive prefix, PROMAL will read the volume name from the selected drive and use it for the volume name part of the file name. It does not change the current prefix.

The following are examples of legal PROMAL file names on the Apple II:

```
AB          1:x7        MyData        YOUR45.T      DOIT.C
HelloThere.C            T12345.S      0:ZZ.SYSTEM   RECOVER.S
2:/WORK1/STUFF          /USER.DISK/MATH/DATA.D      X
```

The names above which do not have a file extension will be stored on disk with the file extension ".C". Thus if you type AB for a file name, the file AB.C will be the file acted upon (EXCEPTION: The EDITOR and COMPILER will assume a file extension of .S for the source files they operate on). The following names are ILLEGAL on the Apple II for the reasons noted in parentheses:

7THDATA.S	(must start with an alphabetic character)
0:MY_Data	(can't have embedded underline, blanks or punctuation)
OLD.STUFF.D	("." can only be used to start the file extension)

EXECUTIVE commands normally use PROMAL file names. However, you may access any file name which is legal with Apple II ProDOS by enclosing the name in quotes. Placing the name in quotes suppresses the default file extension. This allows you to use the EXECUTIVE to copy, delete, and rename files created by BASIC, word processors, or other non-PROMAL programs. For example:

```
COPY "PRODOS"
```

is an EXECUTIVE command to copy the PRODOS system file (not PRODOS.C).

FILE EXTENSIONS (ALL COMPUTERS)

Table 3 below lists the customary file extensions used for various kinds of PROMAL files. Other extensions may be devised by the user for special needs.

TABLE 3

PROMAL FILE EXTENSIONS

Extension	Type of file indicated
.C	A Command file. An executable (compiled) PROMAL program. This is the default extension.
.D	A data file.
.E	A PROMAL EXPORT file (for separate compilation, described in Chapter 8 of the LANGUAGE MANUAL).
.J	A "Job" file, usually prepared with the EDITOR, used to drive the EXECUTIVE from a script of commands.
.L	A program listing
.R	A Commodore 64 relative file (see Appendix **M**)
.S	A PROMAL source program, normally prepared by the EDITOR.
.T	A text file, other than a PROMAL source program.
.X	A cross-reference map (output from XREF utility).

NUMERIC ARGUMENTS

Some EXECUTIVE commands accept numbers for arguments. The built-in EXECUTIVE commands all require numeric arguments to be specified in **hexadecimal.** User-defined programs may specify decimal or hexadecimal at the discretion of the programmer. It is not necessary to understand hexadecimal numbers for casual use of the EXECUTIVE. The Commodore and Apple Reference Manuals describe hexadecimal notation. Hex numbers may be specified with any number of digits; however, the value must not exceed FFFF. In the PROMAL manuals, numbers appearing in the text with a $ prefix indicate a hexadecimal number.

You do not use the $ prefix for EXECUTIVE commands, because the EXECUTIVE implicitly expects hex values.

Normally, blanks are treated as separators between arguments. If you wish to specify an argument which contains an embedded blank, the argument must be enclosed in quotes (either " or '). The EXECUTIVE will then treat the entire quoted string as one argument. It will remove the quotes before acting on the argument. Also, the EXECUTIVE normally "folds" all lower case letters in arguments to upper case before acting on the argument. However, if the argument is enclosed in quotes, no conversion takes place. For example:

```
SET 5000 "Now we will learn PROMAL"
```

will treat the entire quoted character string as one argument, and will not convert it to upper case (this command installs the string specified into memory starting at location 5000 hex).

DEVICES

The PROMAL EXECUTIVE (as well as PROMAL programs) can perform input and output to certain devices as well as files. PROMAL devices are named with a single character, as shown in **Table 4** below.

TABLE 4

DEVICE NAMES FOR PROMAL

Name	Meaning
S	The **S**creen. For output only.
K	The **K**eyboard. For input only.
P	The **P**rinter. For output only.
N	The **N**ull device (discards all output). For output only.
W	The **W**orkspace (in-memory file). For input or output.
L	The **L**ibrary (in-memory file). Normally for input.
T	The **T**elephone (modem). For input/output.

Most EXECUTIVE commands can accept one of these device names anywhere a file can be specified. For example:

```
TYPE L
```

will type the contents of the library on the display.

The W device is a simulated file in memory, also called the Workspace. Although the Workspace is small compared to a disk, it is much faster. The Workspace is often used as a place to save a source program you are working on temporarily. The size of the Workspace can be varied on the Commodore 64 by the WS command. Naturally, if you turn off the computer or leave PROMAL, the contents of the workspace are lost. You can save the Workspace on disk with the COPY command.

The N device simply discards whatever output it receives. This may sound useless but is sometimes useful, as you will shortly see after we discuss I/O redirection.

I/O REDIRECTION

Most EXECUTIVE commands normally output to the screen. However, output may be **redirected** to a file or device by using the redirection operator ">" after the last argument. For example:

```
    TYPE MYLETTER.T >P
```

will type the file MYLETTER.T on the printer instead of the screen. Similarly the command

```
    FILES >W
```

will output the names of all the files on the disk to the Workspace. You may also redirect output to a file, for example:

```
    DUMP 4000 4100 >MEMDUMP.T
```

will output the memory contents to the text file MEMDUMP.T instead of to the screen as it normally would. You can only redirect output to one device at a time.

Many PROMAL programs also allow their output to be redirected in the same manner. Suppose that you had an application program which produced verbose output on the screen each time it ran, and that you wanted to run it without seeing any output. You can do this by redirecting output to the N device.

Some PROMAL programs also allow input redirection. In this case, the program normally accepts input from the keyboard, but can alternatively accept input from a file or device. To redirect input, the input redirection symbol (<) should be used after the last argument. For example, suppose a PROMAL program called INVENTORY normally accepts input from the keyboard and generates a file specified as the first argument. You might tell the program to take its input from another file called APRILINPUT.T instead like this:

```
    INVENTORY APRIL.D <APRILINPUT.T
```

The programming techniques for interfacing to the EXECUTIVE are described in the PROMAL LANGUAGE MANUAL, and make it quite simple to support this kind of command.

PROMAL EXECUTIVE COMMAND SUMMARY

On the following pages are descriptions of the individual EXECUTIVE commands. Commands are presented with a syntax definition, a description of the command's function and examples of use. Unless otherwise noted, the commands are available on both Apple and Commodore 64.

BUFFERS -- Specify Number of ProDOS Disk Buffers -- BUFFERS

AVAILABLE ON APPLE II ONLY.

BUFFERS [Number][HIRES]

The Apple II ProDOS operating system requires that a 1024 byte buffer be allocated in memory for each open file. The BUFFERS command is used to set or display the number of buffers you wish to have set aside. The default is three, which allows three open files at once. Typing BUFFERS without any arguments displays the number of buffers presently assigned. Typing a number after BUFFERS will set the number of buffers specified. Setting the number of buffers will cause all programs to be unloaded, and will affect the memory map. PROMAL allocates the buffers below the available program space.

You also use the argument HIRES (alone or in combination with a number). This will cause PROMAL to unload all programs in memory and reserve space for the hi-resolution graphics "page" from $2000 to $3FFF. This will be necessary before running any program which uses Apple Hi-Res Graphics. You can deallocate the hi-res buffer by typing BUFFERS with a number but without the HIRES argument.

Example 1:

```
BUFFERS
```

will display the current number of 1024 buffers reserved, for example:

```
NUMBER OF BUFFERS = 3
```

Example 2:

```
BUFFERS 1 HIRES
```

allocates one 1024 file buffer plus an 8K byte hi-res screen at $2000. This will of course reduce the size of a PROMAL program which can be loaded.

Notes:

1. You can include a BUFFERS command in a JOB file without harm, even though the buffers for the file may move, so long as there is a buffer available for the job file when the command is completed.

2. Reducing the number of disk buffers below three may adversely affect the operation of the PROMAL COMPILER, since it may need up to three files at once.

3. There is not enough free memory to COMPILE a program after a BUFFERS HIRES command. Therefore when developing a graphics application, remember to switch back to normal mode with a BUFFERS 3 command before compiling.

COLOR -- Change text and background color -- COLOR

AVAILABLE ON COMMODORE 64 ONLY.

COLOR Number [Bkgndnum]
   or
COLOR Colorname [Bkgndcolorname]

The COLOR command is used to change the color used to display text on the screen, and optionally to change the background color. The first argument is the name or number of the desired color for the text, chosen from the following:

0	BLACK	4	PURPLE	8	ORANGE	C	GRAY2
1	WHITE	5	GREEN	9	BROWN	D	LTGREEN
2	RED	6	BLUE	A	LTRED	E	LTBLUE
3	CYAN	7	YELLOW	B	GRAY1	F	GRAY3

The second argument is optional. If specified, it selects the background color. If not specified, the background color is unchanged. Naturally, if you select the same foreground and background color, the text will be invisible (but the computer will still "see" what you type). If you specify a number greater than $F, the EXECUTIVE will force it into the range of 0 to $F by discarding all but the low order 4 bits. Names must be spelled exactly as shown above in order to be recognized (for example, GRAY2 is okay but GRAY 2 is not, nor is GREY2).

Example 1:

```
COLOR PURPLE
```

selects purple characters from this point on.

Example 2:

```
COLOR A 0
```

selects light red characters on a black background.

COPY -- Copy file or device -- COPY

COPY Filename
   or
COPY Source Dest
   or
COPY Filename Prefix

The COPY command is used to copy the contents of a file (or device) to another file or device.

COPY for Apple II

If only one argument is given, it must be a file name, not a device name. It may have a prefix or drive designator specified. The destination for the copy of the file is decided as follows: If a prefix was specified and differs from the current prefix, then the file is copied from the specified prefix to the current prefix. If no prefix was specified, the copy is made from the present drive to the "other" drive (1 to 2 or 2 to 1), if it exists and is ready; otherwise, you will be prompted to swap diskettes for a single drive copy.

In the second form, the **Source** and **Dest** may be file names or device names. The copy is made from the Source to the Destination. The destination file name may be different.

In the third form, the first argument must be a file name, not a device name, and the second argument must be a prefix or drive designator. The file will be copied to the specified prefix with the same name.

For all file names, default file extensions will be applied except for arguments in quotes or devices.

Examples for Apple:

If command is...	and...	Then...
COPY MYFILE	Single drive	will copy MYFILE.C to another disk, prompting for disk changes.
COPY MYFILE	2 drives, current prefix on drive 1	copies MYFILE.C from drive 1 to 2.
COPY 1:MYFILE	current prefix is /RAM/ (RAMdisk)	copies MYFILE.C from drive 1 to /RAM/.
COPY MYFILE.T YOUR.T	anything	copies MYFILE.T to YOUR.T on the current prefix.
COPY PROG.S W	anything	copies PROG.S from the current prefix to the Workspace.
COPY 1:PROG.S 2:	anything	copy file PROG.S from drive 1 to drive 2.
COPY MINE.S YOURS	anything	copy file MINE.S to file **YOURS.C** on the current prefix.
COPY L S	anything	copy the L device (library) to the screen.
COPY COMPILE 0:	anything	copy COMPILE.C from the current prefix to the /RAM/ disk
COPY "PRODOS" 2:	anything	copy file PRODOS from the current prefix to drive 2 and **change its name to PRODOS.C.**
COPY "PRODOS" "2:"	anything	copy file PRODOS from the current prefix to drive 2 with the same name.
COPY /MY.LET/JOE.T	current prefix is /TEMP/	copy JOE.T from prefix /MY.LET/ to /TEMP/ with the same name.
COPY 2:PROG.S P	anything	copy file PROG.S on drive 2 to the printer.

The COPY command can only copy one file at a time and does not support wildcards. However, Apple PROMAL has a utility program, **EXTCOPY**, which can copy multiple files using wildcards (* and ?). It's syntax is:

```
   EXTCOPY Pattern Prefix
```

where Pattern is the desired Filename pattern with wildcards, and Prefix is the destination prefix.

Example:

```
   EXTCOPY *.C 0:
```

copies all files ending in ".C" from the current prefix to the /RAM disk.

COPY for Commodore 64

If only one argument is given, it must be a file name, not a device name. It may have not have a drive designator specified. You will be prompted to swap diskettes for a single drive copy.

In the second form, the **Source** and **Dest** may be file names or device names. The copy is made from the Source to the Destination. The destination file name may be different. Drive designators (0: or 1:) may be used.

In the third form, the first argument must be a file name, not a device name, and the second argument must be a drive designator. The file will be copied to the specified drive with the same name.

For all file names, default file extensions will be applied and a file type of SEQ will be used, except for arguments in quotes. For names in quotes, no default file extension will be applied, and you may specify a file type explicitly by appending a comma and a letter (S for SEQ, P for PRG, or U for USR) to the name inside the quotes (for example, "Basic Prog,P". When enclosed in quotes, names are case-sensitive.

Examples for Commodore:

If command is...	Then...
COPY COMPILE	will copy COMPILE.C to another disk, prompting for disk changes (single drive copy).
COPY MYFILE YOURFILE	will copy MYFILE.C to YOURFILE.C on the same disk.
COPY SOURCE.S	copy SOURCE.S to another disk, prompting for disk changes (single drive copy)
COPY SOURCE.S W	Copy file SOURCE.S to the Workspace.
COPY W S	Copy the workspace to the screen.
COPY MYFILE 1:	Copy file MYFILE.C from drive 0 to drive 1.
COPY 1:TEST.S 0:OLDTEST	Copy file TEST.S from drive 1 to drive 0 with a name change to **OLDTEST.C**
COPY 1:TEST.S 0:OLDTEST.S	Copy file TEST.S from drive 1 to drive 0 with a name change to OLDTEST.S
COPY TEST.L P	Copy file TEST.L to the printer.
COPY "PROMAL,P"	Copy file PROMAL of type PRG to another disk, prompting for disk changes (1 drive)

COPY for both computers

Single drive copies can be made for files up to 64K bytes long. Large files may require several disk swaps. Other copies are limited only by the available disk or device space.

If the file already exists on the destination diskette, you will be prompted:

```
FILE XXXXX EXISTS.
APPEND, REPLACE, OR CANCEL (A/R/C)? _
```

Press the A key to append onto the end of the existing file, R to replace the existing file, or C to cancel the command.

Notes:

1. You will not be able to append the Workspace; copying to it discards any previous contents.
2. Do not attempt to execute a COPY command from a JOB file on disk in a single drive system if the copy will require disk swaps, because PROMAL will attempt to read the JOB file while your destination disk is in the drive. You may execute such a job file by first copying the job file to the Workspace and then using a JOB W command.
3. See **Appendix N** for Commodore dual drive installation instructions.
4. Do not attempt to copy Commodore 64 relative files.
5. On the Apple, attempting to replace a locked file will give a WRITE PROTECTED error message.
6. Attempting to append a locked file or file on a write protected disk will produce a DISK ERROR message.
7. Copying a file to the W device which is larger than the workspace will result in a DISK/DEVICE FULL error with the rest of the file not copied.
8. On the Apple, copying a file from one drive to another when both drives have the same volume name will give an ILLEGAL FILE/DEVICE NAME error.
9. The copy command uses the free memory space for a copy buffer. Therefore you can increase the efficiency of copy operations (especially for single drive copies) by UNLOADing memory before copying.

CS -- Clear screen and home cursor -- CS

CS

The CS command clears the screen and moves the cursor to the home position. It does not have any arguments.

Example:

```
CS
```

DATE -- Display and set date -- DATE

DATE

The DATE command prompts for the current date. It is the same command that is executed automatically when the EXECUTIVE signs on. If you have already entered the date, it will display the current date and then ask you for the desired date. You may enter the date or simply reply with RETURN to keep the present date. The PROMAL compiler uses this date to "stamp" all compiled programs with their creation date. This creation date is displayed by the MAP command when programs are in memory, or by the SIZE command if they are on disk.

Example:

```
DATE

    Today is 9/30/86
    Please enter today's date: _
```

Note:
1. On the Apple IIe, the date will be updated automatically on systems with a Thunderclock or equivalent card when the DATE command is executed, without any prompt or user action.
2. On the Commodore 64, DATE is not a built-in command, but a separate program, which is automatically unloaded after it executes. Therefore you will need to have the file DATE.C on your boot disk to have the DATE command run.
3. If no BOOTSCRIPT.J is present on the boot disk, the PROMAL EXECUTIVE will execute DATE automatically during bootup. If a BOOTSCRIPT.J file is present, DATE will not be automatically executed, so you should have a DATE command in your BOOTSCRIPT.J file unless you don't want to set the date (see JOB for more information).

DELETE -- Delete file from directory -- DELETE

DELETE Filename [...]

The DELETE command removes one or more specified files from the disk directory. The file name(s) to be deleted should be specified as the argument(s). Wildcards are **not** permitted in the Filename. You may not delete a device, only a file name. However, typing DELETE W will clear the workspace. Attempting to DELETE a file on a write-protected disk (or a locked file on an Apple II) will produce an error message.

CAUTION: There is no prompt for verification before the file is deleted, nor is there any way to "un-delete" a file, so use DELETE with caution. You should maintain backups of all important files (for any system, not just PROMAL).

Example:

```
    DELETE MYPROG
```

deletes the file MYPROG.C from the diskette.

You may delete files with no file extensions by specifying the file name in quotes, for example:

```
DELETE "PRODOS"
```

For Non-PROMAL files on the **Commodore 64**, file names enclosed in quotes are case sensitive. For example **delete "promal"** will not delete the file PROMAL but **delete "PROMAL"** will. PRG and USR type files may be deleted.

Notes:

1. DELETE only removes a file from the disk directory. If the deleted file is a program which is also in memory, it is not removed from memory and can still be executed. The UNLOAD command removes the memory-resident program.
2. On the Apple II, you may delete subdirectories, provided they are empty. Remember to enclose the name in quotes.

DISKCMD — Access disk command/error channel — DISKCMD

AVAILABLE ONLY ON THE COMMODORE 64

DISKCMD [Command]

The DISKCMD can be used to send a command to the Commodore disk command channel, and to display the status message from the disk error channel. For normal operations, this command is never needed, since PROMAL and the EXECUTIVE normally handle all command/error processing transparently. You should use this command only for special circumstances requiring special disk commands. If no argument is given, the current disk status will be displayed, for example:

```
00, OK,00,00
```

If the argument is specified, it should be a Commodore disk command enclosed in quotes.

Example:

The following command will **immediately** format the disk currently in the drive, erasing everything currently on the disk:

```
DISKCMD "NO:WORKDISK1,K6"
```

CAUTION: Be very sure you really want to format the disk before typing this command, there is no chance to change your mind!

The example above formats drive 0 (device 8) with a disk name of WORKDISK1 and a disk ID of K6. You should always use a different two letter ID for every disk you format.

A good use of DISKCMD is to issue a "I0" command if you have changed diskettes and think that the new and old diskette may have the same ID. Refer to the Commodore 1541 or 1571 manual for more information on commands which may be issued to the command channel.

DYNO — Enable or disable double speed disk — DYNO

AVAILABLE ONLY ON THE COMMODORE 64

DYNO [Onoff]

PROMAL for the Commodore 64 has DYNODISK, a built-in software package which effectively **doubles the read speed** from Commodore 1541 or 1571 disk drives. Needless to say, this is a tremendous asset, since the Commodore disk is notoriously slow. DYNODISK works with **all** files, not just programs. The DYNO command with no arguments specified will display the current DYNODISK status, either ON or OFF. The optional command **Onoff** must be either the word ON or OFF, and turns the DYNODISK feature on or off.

Examples:

```
   DYNO
```

will display the current status, as either ON or OFF.

```
   DYNO OFF
```

disables DYNODISK for further operations until turned back on. With DYNO OFF, the 1541 will operate at normal speed.

Notes:

1. DYNODISK cannot be used with MSD drives or other drives which are not 100% compatible with the 1541. You can permanently disable dynodisk by setting the byte at $0DE2 non-zero. Do not enable DYNODISK with Skyles FLASH or with other commercial disk speed up cartridges or software. Most other commercial disk speedup products do not work with PROMAL at all because they use some of the same memory needed by PROMAL.

2. Like any 1541 speedup package, DYNODISK does have its drawbacks. **If you have a printer or other device on the serial bus, it must be turned off while DYNODISK is ON.** Failure to observe this rule may hang up the system, requiring a re-boot of PROMAL. If you have a printer with an interface cartridge, it must also be off while DYNODISK is on. Only a single drive is supported. Having DYNODISK on reduces the number of buffers which are available inside the 1541 drive. Therefore you may be able to have fewer open files while DYNODISK is enabled. DYNODISK disables interrupts temporarily while it is reading from disk. Therefore you will not be able to use CTRL-STOP to abort a program while it is reading the disk with DYNODISK on.

3. DYNODISK only doubles the transfer rate of information from the disk drive to the computer. It does not improve the access time or time needed to open a file. Therefore you will observe the biggest speed reductions when reading large files, and little or no improvement on small files. For reads of very small amounts of information, having DYNODISK enabled may actually take slightly longer due to a small setup overhead required. DYNODISK only affects reading speed, not writing speed. See GETBLKF in the LIBRARY manual for programming considerations using DYNODISK.

4. On a 1571 drive, do not send the disk commands to enable the double-sided mode or any of the faster, non-C64-compatible disk modes, with or without DYNO on.

DUMP -- Display memory in Hex and ASCII -- DUMP

DUMP Address [ToAddress]

The DUMP command is used to display a region of memory on the screen in hexadecimal and in ASCII characters. The first argument is the desired starting address. The second, optional argument is an ending address. If no second address is specified, eight bytes will be displayed.

Examples:

```
DUMP 10A0
```

will display eight bytes of memory beginning at $10A0. The display will appear similar to this:

```
10A0   50 0C 4C D6 29 4B 00 52   P.L.)K.R
```

The starting address is shown in the leftmost column. The next eight 2-digit groups show the hex values of $10A0 through $10A7. The rightmost column shows eight characters with the ASCII character equivalent of each of the bytes. Bytes which don't have a printable ASCII character (including blanks) are displayed as "." instead.

```
DUMP 4000 4100 >P
```

will dump the contents of $4000 through $4100 to the printer.

Note: When an end address is specified, a complete line will always be displayed even if it "overshoots" the final address. Therefore the above example will actually display $4000 through $4107, since $4100 will fall in the first position of the last line of the output.

EDIT -- Invoke PROMAL Editor -- EDIT

EDIT [Filename]

The EDIT command is used to invoke the PROMAL full-screen EDITOR. The optional argument is the name of the file to be edited. If no file is specified, the Workspace will be used. If a file is specified and exists, it will be edited. Otherwise, it will be assumed to be a new file to be generated.

The EDITOR is described separately in the EDITOR section of this manual.

Example:

```
    EDIT MYPROG.S
```

edits the file MYPROG.S

Note for **Commodore 64** systems: Normally the EDITOR is always resident in memory and will start immediately when EDIT is typed. However, it is possible to unload the EDITOR from memory by using the "B" option when running the COMPILER to free up extra room for the symbol table (see the COMPILER section of this manual). In this case the EDITOR will automatically be reloaded from disk when needed. The Editor will also be reloaded from disk if the EDITOR program has been corrupted (for example, by an errant user program poking around in memory). You may EDIT larger files by UNLOADing programs or clearing or reducing the Workspace before starting the Editor.

FILES -- Display diskette directory -- FILES

FILES
  or
FILES Pattern* **COMMODORE 64 ONLY FOR THIS FORM**
  or
FILES Subdirectory **APPLE II ONLY FOR THIS FORM**

The FILES command is used to display the names of files on disk.

FILES for the Apple II:

If no argument is given, all files on the current prefix will be displayed, in four columns. If an argument is given, it should be the desired prefix. No wildcards are supported.

Examples:

```
   FILES                    Displays all filenames in current prefix
   FILES 2:                 Displays all file names on drive 2
   FILES /USER.DISK/TEMP/   Displays all files in specified directory
   FILES 0: >P              Prints the names of all files on the /RAM disk.
```

Note:

1. Redirecting FILES to the printer prints one file name per line.

For a more detailed listing of the directory with file sizes, dates, and other information on the Apple, use the **EXTDIR** command. The EXTDIR command is not a built-in command, but a compiled PROMAL program. It requires an argument specifying a pattern to match. The wild cards are * and ?.

Examples for Apple:

EXTDIR *	Displays all files on the current prefix
EXTDIR 2:	Displays all files on drive 2
EXTDIR /TEMP/T*.C	Displays all files starting with T with a .C extension in the /TEMP/ directory.

You will need file EXTDIR.C in order to use the EXTDIR command.

FILES for the Commodore 64:

The files will be listed on the screen in the same format as for BASIC, including the file sizes. Wild cards may be used to display only selected files. The wildcard characters operate exactly as described in the Commodore 1541 Disk Manual. If no Pattern is specified, all the files on the disk will be displayed. The size of each file is measured in Commodore Blocks, which are 256 bytes each. The number displayed is in decimal, not hex. Displaying files does not affect programs in memory. The wildcards allowed are * and ?. The * character matches any string, and the ? character matches any single character.

Examples for Commodore 64:

FILES	Displays all files on drive 0
FILES PR*	Displays all files starting with "PR"
FILES 1:	Displays all files on drive 1
FILES >P	Prints the names of all files

Note:

1. You might suppose it would be possible to display all file names with the extension ".S" by using the "FILES *.S" command. Unfortunately, the Commodore ROMs in the disk drive do not interpret the * wildcard this way. Instead, the *.S pattern is interpreted as matching all files on the disk, no matter what the names are. Therefore the * wildcard is only useful for matching names with a common prefix which differ only in the suffix. The * wildcard cannot be used to represent a common suffix.

2. Redirecting FILES to the printer prints one file name per line.

FILL -- Fill memory area with constant -- FILL

FILL From To Data

The FILL command is used to fill a region of memory with a constant. The first argument, From, is the starting address. The second argument, To, is the final address to fill. The third argument, Data, is the byte to fill with. All arguments are normally specified in hexadecimal. However, the Data argument may be specified as a character in single quotes (') if desired.

CAUTION: If you want to experiment with the FILL command, specify an unused area of memory (shown as "FREE SPACE" by the MAP command). Indiscriminate use of the FILL command could overwrite important programs (such as PROMAL itself) in memory.

Examples:

```
    FILL 4800 4923 0
```

fills $4800 through $4923 with $00.

```
    FILL 4210 4310 ' '
```

fills $4210 through $4310 with ASCII blanks ($20).

FKEY -- Display or change function keys -- FKEY

FKEY [Keynumber String]

The FKEY command is used to display or change the meaning of the function keys, F1 through F8. If no arguments are given, the current function key definitions for all function keys will be displayed. If the optional arguments are specified, then the first argument is the desired function key number to change, 1 through 8. The second argument is the desired character string to be substituted when the function key is pressed. The String may be up to 31 characters long. If it contains blanks, then it should be enclosed in quotes. Only normal, printable characters can be included in the String. Control characters (such as RETURN) cannot be embedded in the String.

Examples:

```
    FKEY
```

will display the current function key definitions. A typical display would be:

Commodore 64	Apple II
F1 = EDIT	F1 = EDIT
F2 = DUMP	F2 = PREFIX *
F3 = COMPILE	F3 = COMPILE
F4 = GET	F4 = GET
F5 = FILES	F5 = FILES
F6 = MAP	F6 = EXTDIR *
F7 = HELP	F7 = HELP
F8 = COPY	F8 = COPY

This is the default list of function key definitions at power-up.

```
    FKEY 2 MYPROGRAM
```

changes function key F2 to generate "MYPROGRAM" when it is pressed.

```
    FKEY 5 "COMPILE MYPROGRAM O=NEWVERSION"
```

changes F5 to be "COMPILE MYPROGRAM O=NEWVERSION".

If you wish to have certain function keys automatically defined when you "boot up" PROMAL, you may do so by simply including the appropriate FKEY commands in the BOOTSCRIPT.J file on your working diskette.

GET -- Load program from diskette -- GET

GET Progname

The GET command is used to load a PROMAL or machine language program into memory without executing it. The argument, **Program,** is the desired file name to be loaded. Normally it is a legal PROMAL file name, written **without** quotes. Only compiled PROMAL programs (or relocatable machine language programs in the PROMAL format as described in Appendix I) can be loaded using this form. An attempt to GET a file of another type will result in an error message. After the program is loaded, it will appear on the MAP display, and can be executed immediately by typing its name.

Alternatively, the Program can be the name of any non-relocatable Machine Language program, enclosed in quotes ("). In this case, the named machine language program will be loaded into memory at the address from which it was saved. No checking is done for overlapping of other programs, nor is space allocated for the program in the MAP. See Chapter 6, function MLGET in the LIBRARY MANUAL, and Appendix I for more details.

Examples:

```
    GET SORT
```

loads the PROMAL program SORT.C into memory (because .C is the default extension). The actual load location will be determined by the LOADer, and can be displayed using the MAP command.

```
    GET "MLSUBS"
```

loads the non-relocatable machine language file called "MLSUBS" into memory at its formerly saved address. It will not show up in the MAP display.

Notes:

1. PROMAL allows several programs to be resident in memory at once, subject to the amount of free memory left. The LOADer will relocate the PROMAL program to an available location. If there is not enough room left, the LOADer will unload programs one at a time, beginning with the last-loaded program, until there is enough room. When the program is loaded, the LOADer also computes and saves a "checksum" of the program image in memory. When you subsequently execute the program, the LOADer will re-compute the checksum and compare it to the saved value. If the two values differ, it indicates that the program in memory has been corrupted (by another program), and so the LOADer will re-load the program from disk before executing it.

2. On the **Apple II,** most machine language programs load initially at $2000 and then "relocate" themselves to their final location. In order to load such a program with the GET command, you need to do a BUFFERS HIRES command first to free the space from $2000 to $3FFF. You also need to insure that the program will not have any other memory "collisions" with PROMAL, including its "relocated" destination, zero page usage, etc.

3. On the **Apple II,** if you GET a PROMAL program, and no error is issued, but when you use the MAP command it does not appear, it means that your program was successfully loaded, but was immediately unloaded when the EXECUTIVE was swapped back in. You will need to UNLOAD. If the symptom persists, then the program is too large to fit in memory at the same time as the EXECUTIVE; you can execute it by typing its name, but you can't GET it from the EXECUTIVE. The COMPILE program exhibits this characteristic.

GO -- Execute Machine Language program -- GO

GO Address

The GO command is used to execute a machine language program already in memory (not a PROMAL program). The argument specifies the starting address for execution in hex. The GO command cannot be used to execute a PROMAL program.

Example:

```
    GO FF81
```

will execute the machine language program at address $FF81. On the Commodore 64, this address is the Commodore 64 Kernal routine "CINT" which re-initializes the screen and video chip.

Notes:

The machine language routine is entered via a 6502 JSR instruction. If the machine language program exits with an RTS instruction, it will return to the PROMAL EXECUTIVE in the normal way. If a machine language BRK instruction is encountered, control will be returned to the EXECUTIVE with an error message similar to:

```
*** RUNTIME ERROR: M/L BRK HIT
P      A   X   Y   F   S
6B01   A3  B2  11  32  F6
AT $5000
*** PROGRAM ABORTED.
```

This display gives the contents of the 6502 registers at the time of the breakpoint and the address of the PROMAL instruction which called the machine language routine. If a GO command was used to enter the program, it will show the starting address instead.

HELP **-- Display help screen --** **HELP**

HELP

The HELP command displays a single screen of information showing some of the control keys used for editing and a list of the most commonly-needed EXECUTIVE commands.

Example:

```
HELP
```

on the Commodore 64 will display a screen similar to:

```
              PROMAL HELP
CTRL-                    CTRL-
A Upper Alpha On/Off       Delete Char.
B Recall Prior Line      [ CRSR to start
K Clear to End Line      Y CRSR to End
            Partial Command Summary
COLOR [Colorname [Background]]
COMPILE [File [L[=List]][O=Object][B]]
COPY File [Dest.]
DELETE File             Function Keys
DUMP From [To]           F1 = EDIT
EDIT [File]              F2 = DUMP
FILES [Pattern*]         F3 = COMPILE
FILL From To Value       F4 = GET
FKEY [Number String]     F5 = FILES
GET Commandfile          F6 = MAP
JOB File.J               F7 = HELP
MAP                      F8 = COPY
RENAME File Newname
SET Addr. Val [Val...]
Type File
Unload [Command]
```

The Apple II help screen is somewhat different (see **MEET PROMAL!**).

Note: Any function key definitions longer than 8 characters will have only the first 8 characters displayed on the HELP menu. The FKEY command will print the entire definition.

JOB — Execute commands in job file — JOB

JOB File.J

The JOB command allows the EXECUTIVE to accept a list of commands from a file on disk instead of the keyboard. This is sometimes called a "batch" capability. Normally this file of commands is prepared using the PROMAL EDITOR. The EXECUTIVE will read commands from the job file until end of file is reached or an error is encountered. Commands should appear in the job file just as they would be typed from the keyboard. Both built-in and user-defined programs may be executed from the job-file "script".

Example:

```
    JOB SCRIPT1.J
```

will cause the EXECUTIVE to read and execute the list of commands on the file SCRIPT1.J. For example, this file might contain:

```
    FILL 4000 4700 0   ; zero memory segment
    GET "MYMLSUBS"     ; load special machine language subroutines
    MAINPROG           ; load & run my PROMAL program
    DELETE TEMPJUNK    ; get rid of unneeded scratch file
    PROG2 >P           ; run my second program, redirect output to printer
```

The EXECUTIVE will attempt to execute all five of these commands before accepting more commands from the keyboard.

When PROMAL is first booted up, it looks for a special JOB file on your working disk called **BOOTSCRIPT.J.** If it finds this file, it will execute the commands on it before accepting commands from the keyboard. You can EDIT the BOOTSCRIPT.J file to do whatever you want. For example, you may want to change the Commodore screen colors with the COLOR command, change the function key definitions with the FKEY command, or execute a certain program automatically when you start the system.

If no BOOTSCRIPT.J is found, the EXECUTIVE will execute an FKEY command and a DATE command by default, to display the function keys and prompt for the date. If you **do** have a BOOTSCRIPT.J file and want the prompt for the date, you should include the DATE command (and the FKEY command if you want) in the script. For the Commodore 64, you will need to have the file DATE.C on disk.

PROMAL supports another feature which greatly enhances the power of JOB files. You can pass arguments (called macros) to a JOB file which will be substituted for "placeholders" in the script. These "placeholders" consist of a \ character (or a "pounds sterling" character on the C-64) followed immediately by a single digit indicating which argument should be substituted. For example \1 will be replaced by the first argument, \2 by the second argument, etc. The first argument is specified after the file name on the JOB command. An example may clarify all this:

Suppose you prepared a file called DO.J which looked like this:

```
EDIT \1.S
DELETE \1.1
COMPILE \1 L=\1.L
UNLOAD \1
\1 TESTDATA.D
```

Now suppose you use the command:

```
JOB DO.J MYPROG
```

The result will be that PROMAL will execute the following commands:

```
EDIT MYPROG.S
DELETE MYPROG.L
COMPILE MYPROG L=MYPROG.L
ULOAD MYPROG
MYPROG TESTDATA.D
```

Each occurrence of \1 was replaced with the first argument (after the file name DO.J) as the commands were read.

Notes:

1. Comments may be included in the Job file, preceded by ";".
2. The ".J" must be explicitly specified in the JOB command.
3. A JOB command may not appear in a JOB file script, except as the last command in the file. You can make a job file that "loops" by making the last command a JOB command with the same filename.
4. A runtime error or any program which executes the library procedure ABORT will terminate the job file and return control to the keyboard.
5. You may execute a JOB file in the Workspace by typing JOB W. Of course you shouldn't use the Workspace for anything else at the same time.
6. The \ character is the "pounds sterling" (£) key on the Commodore 64.
7. Don't put any command in the JOB file that will require swapping the disk the JOB file is being read from (such as a one-drive copy).
8. The JOB file uses one disk buffer, so you may not be able to execute some file-intensive command from a JOB file which works fine when executed directly from the keyboard. Copying the JOB file to the workspace and executing JOB W may alleviate this problem.
9. See the PAUSE command for a useful command in JOB files.

LOCK -- Set write-protect for file -- **LOCK**

AVAILABLE ON APPLE II ONLY

LOCK Filename [...]

The LOCK command sets the write-protect lock on the specified file names. Locked files cannot be written to, renamed, or deleted until they are unlocked.

Example:

```
LOCK COMPILE
```

will write-protect the file COMPILE.C.

Note: Locking a file only provides protection against inadvertent deletion or modification. It does not protect files from hostile users, other software, or from erasure by formatting a disk, nor does it prevent copying or use of the file. See the ProDOS Technical Reference Manual for more details.

MACRO -- Define Macro String -- **MACRO**

MACRO String [String...]

The MACRO command allows you to define macro substitution strings (in the same manner as is described for the JOB command), which can be used interactively instead of in a JOB file. A macro allows a string to be substituted for a two-character abbreviation in an EXECUTIVE command when it is processed. The two character abbreviation consists of the backslash character, \ (or "pounds sterling" key on the Commodore 64) followed by a number from 1 to 8, indicating which string is to be substituted. The MACRO command defines the substitution strings to be used in subsequent commands. The first argument will replace \1, the second argument \2, etc.

Example:

```
MACRO /DEVEL/MYPROG
```

After this command is issued, then:

```
COMPILE \1
```

will be processed by the EXECUTIVE exactly as though you had typed:

```
COMPILE /DEVEL/MYPROG
```

Example 2:

```
MACRO 2:PROCESSDATA " 10 100 200 300"
```

If you then type:

```
\1\2
```

this will be processed by the EXECUTIVE as:

```
2:PROCESSDATA 10 100 200 300
```

Notes:

1. The MACRO command is particularly handy for defining frequently needed volume or path names on the Apple.
2. Macros are independent of function key definitions and may be used inside function key definitions.
3. Macro strings may be any length which will fit the same line as the MACRO command. They may not be nested.
4. A JOB command or another MACRO command redefines all macros.

MAP — -- Display current memory map -- — MAP

MAP

The MAP command is used to display the load addresses of any programs in memory, and a summary of how memory is currently allocated. The actual format of the MAP display varies somewhat depending on the memory organization of the computer.

Example:

```
MAP
```

will display the current memory map. If no programs have been loaded yet, the map should appear similar to this (the actual addresses may differ):

Apple II

```
OBJECT PROGRAMS  $2900-      (0)
FREE SPACE       $2900-8DFF  (25856)
SHARED VARIABLES $8E00-      (0)
EXEC./EDIT SPACE $6100-8DF8  (11520)
TOTAL SPACE      $2900-8DFF  (25856)

ACTIVE WORKSPACE $1200-      (0)
FREE WORKSPACE   $1200-5AFF  (16688)
```

Commodore 64

```
OBJECT PROGRAMS  $4F00-      (0)
FREE SPACE       $4F00-98FF  (18944)
ACTIVE WORKSPACE $9900-      (0)
FREE WORKSPACE   $9900-A0FF  (2048)
SHARED VARIABLES $A100-      (0)
EXEC./EDIT SPACE $A200-CFFF  (11776)
TOTAL SPACE      $4F00-CFFF  (33024)
```

The numbers in the FROM-TO column are in hexadecimal. The numbers in parentheses are the size in decimal. The meaning of the displayed lines is as follows:

OBJECT PROGRAMS	Indicates the total space allocated for all programs currently loaded in memory (excluding shared variables).
FREE SPACE	Indicates the total space presently available for additional programs and variables. This space can also be used to increase the Commodore Workspace (with the WS command). The available space can be increased by UNLOADing programs, reducing the Workspace, or using a NOREAL command. For the Commodore 64, programs may additionally use the EDITor space if needed (described in Chapter 8 of the PROMAL LANGUAGE MANUAL).
ACTIVE WORKSPACE	Indicates how much of the Workspace is currently in use. A WS CLEAR command will set this to 0. For the Apple II, the Workspace is maintained in auxiliary (banked) memory. On the Commodore 64, the Workspace may move as programs are loaded (the contents are maintained).
FREE WORKSPACE	Indicates the size of the unused portion of the Work-space. The WS command can be used to alter this value.
SHARED VARIABLES	Indicates the space allocated for un-initialized global variables (arrays and reals). These variables are allocated at the top of available memory.
EXEC./EDIT SPACE	This is the space occupied by the EXECUTIVE or the EDITOR, or by a users program or variables if needed. The EXECUTIVE and EDITOR share the same space in memory. Only one or the other (or neither) is present at any given time. For the Commodore 64, when a program is executed, the EXECUTIVE is copied to the RAM under the ROMs and the EDITor (which was under the ROMs) is swapped into the vacated space. On exit, they swap again. For the Apple II, copies of the EXECUTIVE and EDITOR are kept in the auxiliary 64K RAM bank and are copied into the EXEC./EDIT space as needed. See Appendix G and Chapter 8 of the PROMAL LANGUAGE MANUAL for more information.
TOTAL SPACE	This is the maximum amount of space presently allocatable for PROMAL programs and variables, if all programs are unloaded, (and, for the Commodore, if the Workspace is cleared and the EDITor's space is used, as described in Chapter 8 of the PROMAL LANGUAGE MANUAL).

After several programs have been run, the MAP command might produce a display that looked more like this:

Apple II

```
FIND       (PRO.)   5/ 2/85 CHKSUM 8171
  CODE $2900-2AFF, VARIABLES $8D00-8DFF
CALC       (PRO.)   8/ 5/85 CHKSUM 8FAF
  CODE $2B00-30FF, VARIABLES $8C00-8DFF

OBJECT PROGRAMS  $2900-30FF (2048)
FREE SPACE       $3100-8BFF (23296)
SHARED VARIABLES $8C00-8DFF (512)
EXEC./EDIT SPACE $6100-8DFF (11520)
TOTAL SPACE      $2900-8DFF (25856)

ACTIVE WORKSPACE $1200-     (0)
FREE WORKSPACE   $1200-5AFF (18688)
```

Commodore 64

```
FIND       (PRO.)   1/ 6/85 CHKSUM B5E7
  CODE $4F00-50FF, VARIABLES $A000-A0FF
CALC        (PRO.)  5/24/85 CHKSUM AE67
  CODE $5100-56FF, VARIABLES $9F00-A0FF

OBJECT PROGRAMS  $4F00-56FF (2048)
FREE SPACE       $5700-96FF (16384)
ACTIVE WORKSPACE $9700-
FREE WORKSPACE   $9700-9EFF (2048)
SHARED VARIABLES $9F00-A0FF (512)
EXEC./EDIT SPACE $A200-CFFF (11776)
TOTAL SPACE      $4F00-CFFF (33024)
```

This display shows two PROMAL programs present in memory, FIND and CALC. The line starting with the name of the program shows that it is a PROMAL program, the date it was compiled, and the checksum which the EXECUTIVE uses to verify the integrity of the program before execution (described in the GET command).

The line beginning with "CODE" shows the starting and ending address of the executable code, and the address range of the variables. It is normal for programs to have overlapping variables allocated, which is why this area is called "shared variables" in the summary. However, the variables may also be allocated immediately after the code portion, if the program had the keyword OWN on its PROGRAM line when it was compiled (discussed in the PROMAL LANGUAGE MANUAL).

Memory is always allocated in pages of exactly $100 bytes each (256 decimal). This is why all the address ranges start with $xx00 and end with xxFF. Therefore even if a program is only a few bytes long, it will still be allocated a whole page. See Chapter 8 of the LANGUAGE MANUAL for further information. The SIZE command can be used to determine the true size of a program to the exact byte.

For the Commodore 64, notice that the Workspace has moved slightly to make room for the new shared variables. This will not affect the content of the Workspace, which is maintained by the LOADer.

Notes:

1. Memory areas not shown on the MAP display are used by the PROMAL system. A detailed memory map for PROMAL system usage is shown in **Appendix G.** The EDITOR and COMPILER will use all of the area indicated as FREE SPACE for buffers when they run. In addition, they may use the Commodore Workspace if it is completely empty (see the EDITOR section of this MANUAL for a full discussion of when the EDITOR uses the Workspace). The Commodore COMPILER also has an option of using more memory for its symbol table (see the COMPILER section of this manual).

2. If you GET the Apple compiler, it will not show up in the memory map. This occurs because the compiler will be immediately unloaded from memory after it is loaded to make room for the EXECUTIVE, which overlaps it. This is of no particular consequence, except that you must have a copy of the compiler on disk if you want to run it.

NEWDIR — Create a new sub-directory — NEWDIR

AVAILABLE ON APPLE II ONLY

NEWDIR Name

The NEWDIR command is used to create a new subdirectory called Name. The name should be a legal ProDOS sub-directory name. A "/" will be added to the end of the name if one is not specified. If the name does not **start** with a "/" character, then the current specified name will be appended to the current prefix.

Example:

```
NEWDIR MUSIC/     ; Creates new subdirectory /MUSIC/ on current prefix.
```

NOREAL -- Remove real number routines -- NOREAL

NOREAL

The NOREAL command is not normally needed, but allows you to discard all routines from the PROMAL runtime software and resident library which support the processing of REAL (floating point) data. This adds approximately 2.5 K bytes of additional available space for editing, compilations, or other programs. However, you will not be able to successfully execute any program which makes use of REAL data (the EDITOR and EXECUTIVE do not; the COMPILER only uses REALs if the program it is compiling uses REALS).

When the NOREAL command is executed, the EXECUTIVE will first UNLOAD **all programs from memory.** It will then move the bottom of available memory down by about 2.5K. You may then resume normal operations.

If you normally have no need for floating point capabilities (many application areas such as games, music, graphics, and text editing normally don't need REAL data), you may wish to include the NOREAL command in your BOOTSCRIPT.J file (described in the JOB command, above) to automatically free up more space when you boot up.

Once NOREAL has been executed, the only way to restore the REAL processing is to reboot the system.

If you get either of these two messages, it indicates that you may have tried to execute a program which uses REALs after you have discarded REAL processing:

```
*** RUNTIME ERROR: ILLEGAL OPCODE
AT xxxx

*** RUNTIME ERROR: REQ'D PROGRAM NOT LOADED
AT xxxx
```

PAUSE -- Pause in JOB file -- PAUSE

PAUSE ["Message"]

The PAUSE command is normally only used in JOB files. When executed, it displays the text of a message enclosed in quotes and then waits for a carriage return from the keyboard. The text of the message must be enclosed in double quotes.

Example:

```
PAUSE "PRESS RETURN TO CONTINUE."
```

When executed in a JOB file, this will pause and display "PRESS RETURN TO CONTINUE" on the screen. If you press RETURN, the job will continue with the next command. Alternatively, you could press CTRL-STOP (Commodore) or CTRL-C (Apple) to abort the job at this point in the JOB file execution.

PREFIX -- Set or view default pathname -- PREFIX

AVAILABLE ON APPLE II ONLY

PREFIX [Pathname]
or
PREFIX * [Slot Drive]

The PREFIX command is used to set or show the current prefix (path name) which will be used for subsequent file references. The PREFIX command without any arguments displays the current path name. If a name is specified as an argument, it should be a legal path name (starting and ending with / characters). This form is most often used to select a different diskette (ProDOS Volume) for subsequent action. The form PREFIX * will change the pathname to whatever the volume name is for the diskette in the drive which booted up PROMAL, and display this name. This form is most often used after changing diskettes.

Examples:

PREFIX	will display the current path name, for example
PREFIX *	updates the current prefix to the volume in drive 1
PREFIX 2:	sets the current prefix to the volume in drive 2
PREFIX /WORK1/	sets the current prefix to /WORK1/
PREFIX * 6 1	sets the current prefix to slot 6 drive 1

If ProDOS can not find a diskette with the specified volume name, a DEVICE NOT READY error will be given. You can use the PREFIX * command to determine the volume name of an unknown diskette. Volume names are initially assigned when the diskette is formatted.

IMPORTANT: You should always execute a PREFIX * command after changing diskettes. Otherwise, ProDOS will not be able to find your commands or files, because the current prefix will still be the removed volume name.

QUIT -- Quit PROMAL -- QUIT

QUIT

The QUIT command exits from the PROMAL system. The workspace will be lost.

For the **Commodore 64:** QUIT resets the computer back to its power-on status in BASIC. You may **not** re-enter PROMAL except by re-loading it from disk.

For the **Apple II:** QUIT will change to 40 column mode and prompt for the name of a "prefix". Enter the name of the desired system to run, which must end in ".SYSTEM". For example, to run BASIC, you might enter /WORK1/BASIC.SYSTEM.

Example:

```
QUIT
```

It is not necessary to QUIT before turning off your computer. You may simply remove the diskette and power off the computer in the normal manner, provided the disk is not being accessed.

RENAME -- Rename a file -- RENAME

RENAME Oldfile Newfile

The RENAME command is used to rename a file on disk. The first argument is the existing file name and the second is the new file name desired.

Examples:

```
    RENAME MYFILE YOURFILE
```

changes the file named MYFILE.C to YOURFILE.C.

```
    RENAME DATA1.D DATA1.T
```

changes the extension of the file.

For the **Commodore 64:** Non-PROMAL files may be renamed by enclosing the file name (case sensitive) in quotes, for example:

```
   RENAME "BASIC PROG" "OLD_PROG"
```

For the **Apple II:** Any path name can be specified for the old file. If you wish the new name to not have a file extension, enclose it in quotes. File names without extensions should be enclosed in quotes. You may rename a directory (in quotes). Remember to change the PREFIX after you do. You can rename a volume name (in quotes, no trailing '/').

SET -- Set memory address to value -- SET

SET Address Value [...]

The SET command is used to install values in memory. The first argument is the starting address, in hex. The second (and optionally additional) arguments specify the values to be installed into memory in sequence. Each **Value** argument can be:

(1). A byte specified in hexadecimal
(2). A word greater than $00FF specified in hexadecimal
(3). A string to be terminated by a 00 byte, enclosed in double quotes (")
(4). A string enclosed in single quotes (')

The difference between (3) and (4) above is that if the string is enclosed in double quotes ("), a 00 byte terminator will be installed automatically after the last byte of the string, but if it is enclosed in single quotes ('), no 00 byte is added.

CAUTION: Indiscriminate use of the SET command may overwrite important programs or data in memory (such as PROMAL itself). Before experimenting with the SET command, you should pick a "safe" location, such as in the FREE SPACE displayed by the MAP command.

Example:

```
    SET 54B0 4B
```

sets the byte at location $54B0 to $4B.

```
    SET 5700 A921 0 143
```

sets location $5700 to $21, $5701 to $A9, $5702 to $00, $5703 to $43, and $5704 to $01 (note: word values are stored in the standard 6502 processor order with the low-order byte first and the high-order byte at the next address).

```
    SET 4B10 'Hello'
```

installs five bytes starting at $4B10 (lower case letters are not changed to upper case).

```
    SET 4B10 "Bye now"
```

installs the specified string starting at $4B10, and adds a $00 byte after the last character installed in memory (the $00 byte is used to terminate a string as defined by the PROMAL LANGUAGE MANUAL).

SIZE -- Display program size -- SIZE

SIZE Filename

The SIZE command is used to display the memory requirements of a PROMAL compiled program without actually loading it into memory. The argument is the name of a legal PROMAL command file. The EXECUTIVE will read the header from the file and display the pertinent information in a form similar to the MAP command.

Example:

```
    SIZE BILLIARDS
```

will display the load requirements of the BILLIARDS.C file without actually loading the program into memory. The SIZE command display will be similar to:

```
    BILLIARDS   (PRO.) 9/21/84  VER 2
     CODE $06D7, GLOB VARS $0060, $10
```

The first line displays the command name, the kind of program (PROMAL), the compilation date, and the PROMAL version that created the object module (1 for pre-2.0, 2 for version 2.0 and above). The second line indicates the size of the executable program code in hexadecimal bytes, and the amount of memory required for arrays and simple variables, in bytes. No load address is shown, because the EXECUTIVE has not loaded the program.

TYPE -- Display file on screen -- TYPE

TYPE Filename

The TYPE command is used to display the contents of a **text** file or **source** program on the screen. It can also be used to display the contents of the Workspace. Output can be redirected to another PROMAL output device such as the printer. The argument is the name of the file or device to be typed.

Examples:

```
    TYPE FIND.S
```

types the file FIND.S on the screen.

```
    TYPE MYJOB.J >P
```

types the file MYJOB.J on the printer.

```
    TYPE L
```

types the standard library (L device) on the screen.

```
    TYPE K >P
```

turns your computer into an electronic correcting typewriter. All lines typed on the keyboard will be output to the printer when RETURN is typed. To terminate the command type CTRL-Z, which indicates "end of file" from the keyboard.

Notes:

1. If you attempt to type an compiled program or other non-text file, it will display garbage on the screen. This can happen if you forget to specify the ".S" extension on the filename to be typed.

2. On the **Commodore 64**, you may slow a TYPE display down by holding down CTRL. You may pause the display temporarily by holding down STOP. You can abort the command by pressing CTRL/STOP.

3. On the **Apple II**, you may temporarily halt the display with CTRL-S, and continue the display with any key. CTRL-C will abort the command.

UNLOAD -- Unload program from memory -- UNLOAD

UNLOAD [Commandname]

The UNLOAD command is used to remove a command (PROMAL compiled program) from memory, making space available for other programs. This will also make more room available for the EDIT buffer and the COMPILER's tables, which use the available space for temporary storage. The optional argument is the name of the command to be UNLOADED. If the command is not found, no error is indicated. If the command name is found, it will be removed from the EXECUTIVE's internal table of loaded programs, **along with any programs which occupy higher addresses.** If the **Commandname** argument is not specified, all commands are unloaded.

Example:

```
    UNLOAD BILLIARDS
```

removes the BILLIARDS program from the EXECUTIVE's command list, freeing up memory previously allocated to it for other programs. The MAP command can be used to display the results.

Notes:
1. When a program is unloaded, the memory it occupied is not cleared or altered in any way. The space is merely deallocated, making it available for other programs as they are loaded.
2. If two loaded programs have the same name, only the last one loaded will be unloaded. This situation arises when different object file names were loaded which had the same PROGRAM name when compiled.

UNLOCK -- Unlock a file -- UNLOCK

AVAILABLE ON APPLE II ONLY

UNLOCK Filename [...]

The UNLOCK command removes the write-protect status from a file previously LOCKed. Once UNLOCKed, the file can be deleted, renamed, or overwritten. File names without an extension may be unlocked by enclosing the file name in quotes.

Example:

```
UNLOCK MYSOURCE.S
```

WS -- Change or clear Workspace -- WS

```
WS Size      ; This form is available on Commodore 64 only
  or
WS CLEAR     ; This form is available on both computers.
```

Description:

The WS command is used to change the size of the Workspace (the W device in-memory file), or to discard its contents. The first form (for Commodore only) has an argument which specifies the desired Workspace size in hex bytes. If the value specified is less than $20, it is assumed to be in hexadecimal K-bytes instead. If the size requested is larger than the remaining available space, NOT ENOUGH MEMORY will be displayed. If the ACTIVE (in use) Workspace is larger than the requested Workspace, no action takes place and the message "ACTIVE W IS LARGER. (WS CLEAR WILL EMPTY W)" will be displayed.

The second form is used to clear the Workspace. The memory content of the Workspace is not actually altered, but the internal pointers used to manipulate the Workspace are set so that the active Workspace is 0.

Examples:

```
    WS E
```

sets the Commodore Workspace size to 14K bytes ($E times $0400).

```
    WS 1800
```

sets the Commodore Workspace size to $1800 bytes (6K bytes).

```
    WS CLEAR
```

clears the active Workspace to 0 without affecting the size of the Workspace.

Notes for **Commodore 64**:

1. The default size of the Workspace is 2K bytes (2048 decimal). The location of the Workspace varies as programs are loaded and unloaded. The EXECUTIVE maintains the contents of the Workspace as it is moved around in memory. The COMPILER and EDITOR will use the Workspace for buffer space if it is completely empty. Therefore you can EDIT or COMPILE larger programs by issuing a WS CLEAR command first (See the EDITOR section of this manual for a more complete description of the EDITOR's treatment of the Workspace).

Notes for **Apple II**:

1. The workspace is maintained in auxiliary memory (the "other" 64K of banked memory). The Apple has a fixed size workspace of about 18K bytes.

THE PROMAL EDITOR

The PROMAL EDITOR is a full-screen text editor for preparing and changing text files. It incorporates many of the features found in the finest word processors, but is designed specifically for the generation of PROMAL source programs. Some of the important features of the EDITOR are:

* Cursor-driven, full-screen operation.
* Displayed function key legends and on-line HELP screen.
* Automatic vertical scrolling.
* Automatic horizontal line scrolling
* Insert or type-over mode.
* Global search and search-and-replace with "veto".
* Block copy, move, delete, save, and recall.
* Semi-Automatic indentation support for PROMAL programs.
* Fast operation.

The EDITOR is loaded into memory automatically when PROMAL is booted up, and is normally a permanent part of the PROMAL system in memory. This makes it very convenient to use, since the EDITOR is always instantly available.

ENTERING THE EDITOR

From the EXECUTIVE, you may enter the EDITOR with the EDIT command, of the form:

EDIT [Filename]

where **Filename** is an optional argument specifying the name of the file to edit. If the **Filename** is omitted, the EDITOR will use the file in the Workspace, if any. Otherwise, a new file will be assumed. If a **Filename** is specified and does not exist, the EDITOR will start a new file by that name. The EDITOR also assumes a default file extension of ".S", so it is not necessary to specify the extension when entering the file name.

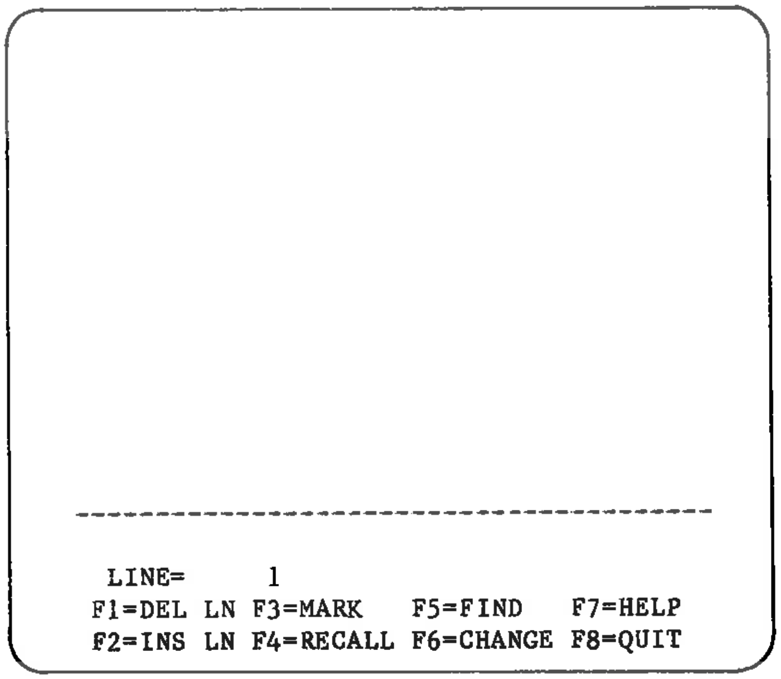
The EDITOR begins by "signing on". If a new file is being created, this display will not be visible long enough to read because the screen will be immediately cleared and the cursor moved to the upper left-hand corner of the screen for you to start your new text file or program. If you are editing an existing file, the message will be visible until the file has been loaded into memory.

THE EDITOR DISPLAY FORMAT

For a new file, the EDITOR will show an initial display as shown in Figure 1a (For Commodore 64) or 1b (for Apple II) below.

FIGURE 1a

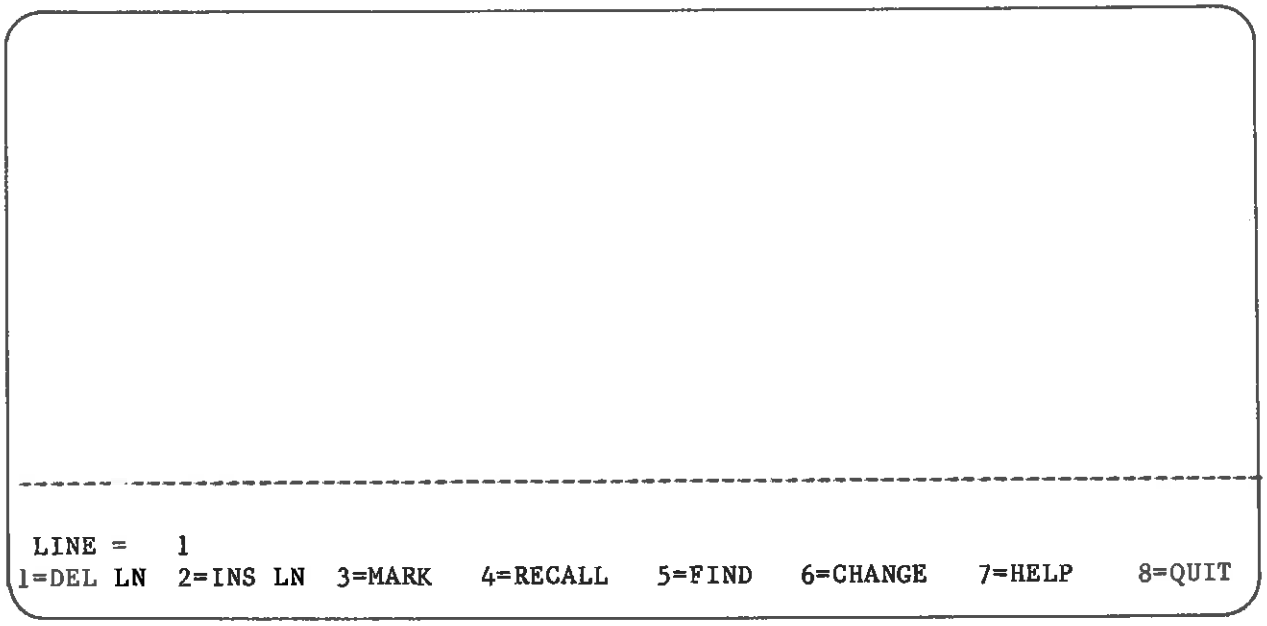
COMMODORE EDITOR SCREEN DISPLAY FOR A NEW FILE



```
 LINE=      1
F1=DEL LN F3=MARK   F5=FIND   F7=HELP
F2=INS LN F4=RECALL F6=CHANGE F8=QUIT
```

FIGURE 1b

APPLE II EDITOR SCREEN DISPLAY FOR A NEW FILE



```
 LINE =   1
1=DEL LN  2=INS LN  3=MARK   4=RECALL   5=FIND   6=CHANGE   7=HELP    8=QUIT
```

The first 20 lines of the screen are initially blank for a newly created file, or contain the first 20 lines of an existing file. This area is called the **text area** and is where most editing operations take place. A full-width dashed line separates the text area from the bottom 4 lines of the screen, called the **status area.** This area displays the meanings of the current function keys and displays status information such as the current line number in the file. The line number display will change automatically as you move about in the file or insert, delete or move lines.

To create text in a new file, just type in the normal fashion. The RETURN key will advance the cursor to the next line. If you reach the end of the text area, the text area will automatically scroll up by one line to make room for additional text. The lines that scroll off the top of the screen are not lost but are merely no longer visible. If you move the cursor up beyond the top line, the text area will scroll down 1 line, until you stop moving the cursor up or reach the beginning of the file.

If the cursor stops advancing as you type and characters overprint one on top of the other, it means that either (1) you have reached the maximum line size so no more characters can be entered on this line, or (2) the edit buffer is completely filled.

In the EDITOR, you have all of the editing keys available to you that were in the EXECUTIVE, plus some additional ones. These keys are summarized in **Table 5** below.

The best way to learn how to use the EDITOR is to EDIT an existing file and experiment with the various keys. You will not alter the existing file on disk unless you specifically elect to overwrite the existing file when you exit from the EDITOR, so you won't do any harm.

EDITOR FUNCTION KEYS

The function keys operate somewhat differently in the EDITOR than in the EXECUTIVE. The meaning of each of the function keys is always shown in the status area of the screen. Under some circumstances, these function key "legends" will change during command processing, to allow more than a total of 8 possible actions. Some of the function keys perform an operation immediately, and some require additional input to be typed followed by RETURN. The action of each of the function keys is described in the following sections.

For the **Apple II,** function keys are activate by holding down either Apple key and pressing the desired number key.

THE HELP DISPLAY

Pressing the F7 (HELP) function key will temporarily erase the screen and display a screen showing the meanings of the control keys. Pressing RETURN will restore the normal display.

TABLE 5

PROMAL EDITING KEYS

Commodore Key	Apple Key	Description
RETURN	RETURN	End of line. Advances to the start of the next line. May be typed at any position in the line.
DEL	DELETE	Replace the character left of the cursor with a blank and backup the cursor one position . Will "pull back" characters to the right only if in "insert mode".
INST	CTRL E	Enable "insert mode". Any characters subsequently typed will be inserted before the character the cursor is on, pushing any existing text to the right. Exit insert mode by pressing RETURN or any cursor key.
CTRL <--	CTRL D	Delete character with pullback. Deletes the character under the cursor and pulls any remaining text to the left to fill in the gap. On the **Commodore 64**, this key is located above the CTRL key.
==>	-->	Cursor right. Moves the cursor to the right, without altering the character under the cursor. Stops at the end of the line. Repeats automatically after a brief pause if held down. On the **Commodore 64**, advancing beyond column 40 will cause the line to temporarily scroll left, allowing editing beyond column 40. On the Apple, line scrolling begins at column 80.
<==	<--	Cursor left. Moves the cursor to the left, without altering the character under the cursor. Stops at column 1 of the line. Repeats automatically after a brief pause if held down.
CRSR up	up arrow	Cursor up. Moves the cursor up by one line. If already at the top of the screen, scrolls the screen down to back up by one line, until the beginning of the file is reached.
CRSR down	down arrow	Cursor down. Moves the cursor down by one line. If already at the bottom of the text area, scrolls the screen up to advance by one line in the file, until end of file is reached. Will not advance beyond end-of-file (use RETURN to add new blank lines at end-of-file).

Note: N/A means not available. CTRL X means hold down the CTRL key and press X.

TABLE 5 (continued)

Commodore Key	Apple Key	Description
HOME	CTRL Y	Position the cursor to the upper left hand corner.
CTRL X	CTRL X	Clear the entire line. Erases all characters typed on the line and repositions the cursor to the first column of the same line.
CTRL K	CTRL \	Clear to end of line. Erases all characters from the cursor to the end of line.
CTRL Y	CTRL L	Jump to last character on line. Moves the cursor to the character after the last character on the line, without affecting the line content.
CTRL [	CTRL F	Jump to first character on line. Moves the cursor to the first character position, without affecting the line content.
CTRL A	CTRL A	Toggle Alpha-Lock mode. Switches in or out of ALPHALOCK mode. If the ALPHALOCK indicator appears in the status area, then all subsequent alphabetic characters to be entered and displayed as upper case when typed. Pressing CTRL A again returns to normal upper and lower case alpha mode.
CTRL I	TAB or CTRL I	Tab and indent. Moves the current level of indentation in by two columns, forming a new temporary margin. Generally used after a conditional statement in a PROMAL program.
CTRL U	CTRL Q	Un-indent. Moves the current left margin two columns left, so that subsequent text will be entered with one less level of indentation. Generally used to end an indented block following a conditional statement in a PROMAL program.
CTRL N	CTRL N	Next page. Erases the text area and displays the next 20 lines from the file being edited.
CTRL P	CTRL P	Previous page. Erases the text area and displays the prior 20 lines from the file being edited.
CTRL J	CTRL ^	Adjust right. Moves the line the cursor is on to the right by one level of indentation (2 columns) and advances the cursor to the next line. Usually used to indent an existing block of statements in a PROMAL program when adding a conditional statement in front of the block.

Note: N/A means not available. CTRL X means hold down CTRL key and press X.

TABLE 5 (continued)

Commodore Key	Apple Key	Description
CTRL O	CTRL O	Adjust left. Moves the line the cursor is on to the left by one level of indentation and advances to the next line. Usually used to remove a level of indentation from a PROMAL program.
CTRL W	CTRL W	Set window. Erases and redisplays the text window starting with the cursor's current column. Usually used to examine or edit the right-hand portion of a group of lines which are wider than the screen. Does not effect the line content. The characters in leftmost column will be highlighted to emphasize that text exists off the left side of the display.
CTRL V	CTRL V	Normalize window. Restores the normal text display position so that lines are displayed beginning with column 1. Used to remove effect of a prior CTRL W.
CTRL B	CTRL B	Backtrack. Erases any existing command line and enters the last EDITOR command entered. More can be typed after the recalled command, or it can be edited further. Pressing CTRL B again will recall the next-to-most recent line entered. This can be repeated up to the limit of the backtrack buffer of 256 characters; then the display will "wrap around" to repeat the most recent command again.

Note: N/A means not available. CTRL X means hold down the CTRL key and press X.

INSERTING AND DELETING LINES

Pressing the F1 (DEL LN) function key will cause the line the cursor is on to be deleted and the lines below it will move up to fill in the gap. If you delete a line accidentally, you can recover it by pressing the F4 (RECALL) function key followed by the RETURN key.

Pressing the F2 (INS LN) function key will open up a new, blank line above the line the cursor is currently on, and move to the start of the new blank line. Existing lines of text are pushed down to make room for the new line.

SEARCHING WITH THE FIND KEY

The F5 (FIND) function key can be used to jump to a specific line number in the file or to locate a particular character string. If you press F5, the word FIND will appear in the status area, followed by a blinking cursor. If you type in a number followed by RETURN, the text area will be immediately redisplayed starting with that line number. For example:

```
    FIND 67
```

will immediately display lines 67 through 86 of the file. You can also jump to the end of the file by merely typing a number greater than the last line of the file, for example FIND 9999. The largest line number that can be entered is 32767. You may jump to the beginning of file using a FIND 1 command.

You may also enter a displacement instead of an absolute line number. For example,

```
    FIND +50
```

will redisplay the text area beginning 50 lines from where you are now, and:

```
    FIND -200
```

will back up 200 lines from where you are now.

You may search for a certain string with the F5 function key by specifying the desired string in quotes. For example:

```
    FIND 'INFILE'
```

searches for the string "INFILE". Either single (') or double (") quotes may be used to enclose the string, but must match on both ends. Searches are **always made in the forward direction, starting from the present cursor position** (not necessarily at the top of the screen). If the string is found, the text area will be redisplayed starting with the line containing the string, with the cursor on the first character of the string. If the string is not found, a "NOT FOUND" message will be displayed in the status area and the text area will not be changed. Press RETURN in this case to remove the message.

You may also combine searches on a single command. For example to jump to the beginning of file and then search for 'WHILE GETC', you could press the F5 function key and complete the command as:

```
    FIND 1 "WHILE GETC"
```

If "WHILE GETC" is not found, then the text area will redisplay starting at the first line after the NOT FOUND message. In a compound search such as this, the screen will display the last "successful" part of the search (that is, the FIND 1 part in this case). **A particularly useful compound search is:**

```
    FIND +1 'XXXXX'
```

where XXXXX is whatever text you are looking for. If this finds an occurrence of XXXXX, but not the one you wanted, you can repeat the search starting with the next line by just pressing CTRL B and RETURN. You can also use a compound search to locate a selected word in a particular subroutine. For example:

```
    FIND 1 'PROC SUB1' 'ARG'
```

will find the first occurrence of 'ARG' in procedure SUB1, but will ignore all prior occurrences in the file, because the EDITOR will first jump to line 1, then search for 'PROC SUB1', and then search forward for 'ARG'.

SEARCH AND REPLACE

Function key F6 (CHANGE) is similar to the FIND function key, but allows you to change a string to a different string automatically. Like the FIND command, you will need to complete the command after pressing the F6 key. For example:

```
    CHANGE 'SPRITEXY  'SPXY'
```

will search forward from the present cursor location for the string "SPRITEXY", and when it is found will highlight the matching string in the text window and display the prompt:

```
    CHANGE THIS STRING (Y/N/C=CANCEL)?_
```

Replying with the Y key will cause the string to be replaced and the command completed. Replying with the N key will cause the EDITOR to search for the next occurrence of the string, and repeat the prompt when it is found. Pressing the C key will abort the command with the message "0 CHANGES MADE".

More often, you will want to use the CHANGE command to replace several or all the occurrences of a certain string. This can be done by replacing a repeat constant in front of the string to be searched for. For example:

```
    CHANGE 999 "SPRITEXY" "SPXY"
```

will tell the EDITOR to replace up to 999 occurrences of the string "SPRITEXY" with "SPXY", again searching forward from the present cursor location. The EDITOR will pause at each occurrence, highlight it, and prompt:

```
    CHANGE THIS STRING (Y/N/C=CANCEL)?_
```

This gives you a chance to "veto" any occurrences which you don't want to change. This is very important, since the EDITOR will often "surprise" you with occurrences you didn't think of. For example, "TEN" may appear in "OFTEN", which you really didn't want to change. After the last occurrence is found, the EDITOR will tell you how many changes were made.

"CUT AND PASTE" OPERATIONS

The PROMAL EDITOR supports a variety of block operations, often known as "cut and paste" operations. A block of text is one or more complete lines of text to be operated on as a group. Blocks can be moved, copied, deleted, saved to a file or inserted from a file. To designate a block, place the cursor in any column of the first line of interest and press the F3 (MARK) key. The line will be highlighted, and the function key legends will change to:

```
F1=DELETE  F3=MARK   F5=FIND  F7=COPY
F2=        F4=WRITE  F6=MOVE  F8=CANCEL
```

which indicate your new choices (shown on one line on the Apple II). Move the cursor to the last line of the text block and press F3 (MARK) again. For large blocks you may need to scroll the screen or use a FIND command to locate the last line desired. All intervening lines will be highlighted when you press the F3 key again. Any number of lines can be marked. You have now designated the block to operate on. The F8 (CANCEL) key can be used at any time to cancel the command. You may also mark the block with the last line first and then the first line; it doesn't matter. If you wish, you can also just press F3 repeatedly until you have the desired number of lines marked. Each depression of F3 will mark one more line.

If you want to move the block or copy it elsewhere in the file, position the cursor to the desired destination and press F7 (COPY) or F6 (MOVE). The marked block will be inserted above the line the cursor is on. The highlighting will be removed, completing the command. Copying a block requires as much free space as copying. Therefore when moving large blocks, you may wish to use WRITE block, DELETE block, RECALL block instead (described below). If you want to delete the marked block, press F1 (DELETE) instead. **CAUTION: THERE IS NO WAY TO "UN-DELETE" A DELETED BLOCK.**

SAVING A BLOCK TO A FILE

If you want to save a marked block to a file, press the F4 (WRITE) key. The word "WRITE" will appear in the status area, followed by the cursor. Complete the command by typing the name of a legal PROMAL file to receive the copy of the marked block. The file will be created and written on disk, and the highlighting removed, completing the command. For example:

```
    WRITE TEMP.S
```

will copy the marked block into the file TEMP.S on disk. (Note: if you omit the extension, the EDITOR will supply a ".S" extension by default.) You may also write to devices. For example:

```
    WRITE P
```

will type the marked block on the printer. You may also write a block to the Workspace, under some circumstances (see the discussion of the workspace and the edit buffer at the end of this section for more information). You cannot append to the Workspace, however. Do not try to write to the S device.

To insert a file on disk into the file being edited, move the cursor to the desired location and press F4 (RECALL). Complete the command by typing in the name of the desired file. The file will be inserted above the present cursor position. To insert a file after the very last line of text, press RETURN to put the cursor on a new line below the last line; then do the insert.

When you start writing your own PROMAL programs, you will find the BLOCK-WRITE and RECALL commands very useful for copying pieces of an existing program into a new program, so you don't have to type all those lines over.

EXITING FROM THE EDITOR

Function key F8 (QUIT) will display the buffer status and exit menu,

for example:

```
          PROMAL EDITOR VERSION 2.1
       COPYRIGHT (C) 1986 SMA, INC.

     FILE NAME = W (AUTO-UPDATE ON QUIT)
     BUFFER SIZE = 10685
     FILE SIZE = 9596


          R = REPLACE ORIGINAL FILE

          N = WRITE TO NEW FILE

          W = WRITE TO WORKSPACE

          C = CONTINUE EDITING

          Q = QUIT EDITOR

     SELECTION?

     _
```

The buffer size and file size are shown in decimal bytes. The buffer size represents the maximum file size that the EDITOR can work on (it can be increased by unloading programs or, on the Commodore 64, by clearing the Workspace).

Press the key corresponding to the desired option and press RETURN. R will replace the original file you specified on entry to the EDITOR, if any (**it may be a new file you specified on entry to the EDITOR, too**). If you select N, you will be prompted to enter the name of the new file. The W option writes to the Workspace (W device). It is possible that the Workspace may not be large enough to hold the entire file edited. In this case, you will be asked if you wish to write to the Workspace anyway, in which case the file will be truncated. Otherwise you can write the whole file to the disk to save it. On the Commodore 64, when "auto-updating" the Workspace, the Workspace may be increased in size automatically if needed to hold the file.

For convenience, the Workspace is **automatically updated** when you Quit the EDITOR if you entered the EDITOR **without** specifying any file or device name. This is so that you will have your corrections saved in the workspace without having to remember to select the W option explicitly. (NOTE: You can edit the workspace without having it automatically updated on exiting from the editor, by starting the EDITOR with the command "EDIT W" instead of just "EDIT").

After you have made one selection, such as writing the file to disk, you can make another selection. The C option will let you continue editing the file. It is a good idea when you are doing a lot of editing to periodically write the file out to disk so all is not lost in the event of a power failure or similar problem. When you are ready to return to the PROMAL EXECUTIVE, enter Q and RETURN. This will clear the screen and return to the EXECUTIVE. NOTE: Never abort the EDITor with CTRL-STOP (Commodore) or CTRL-RESET (Apple).

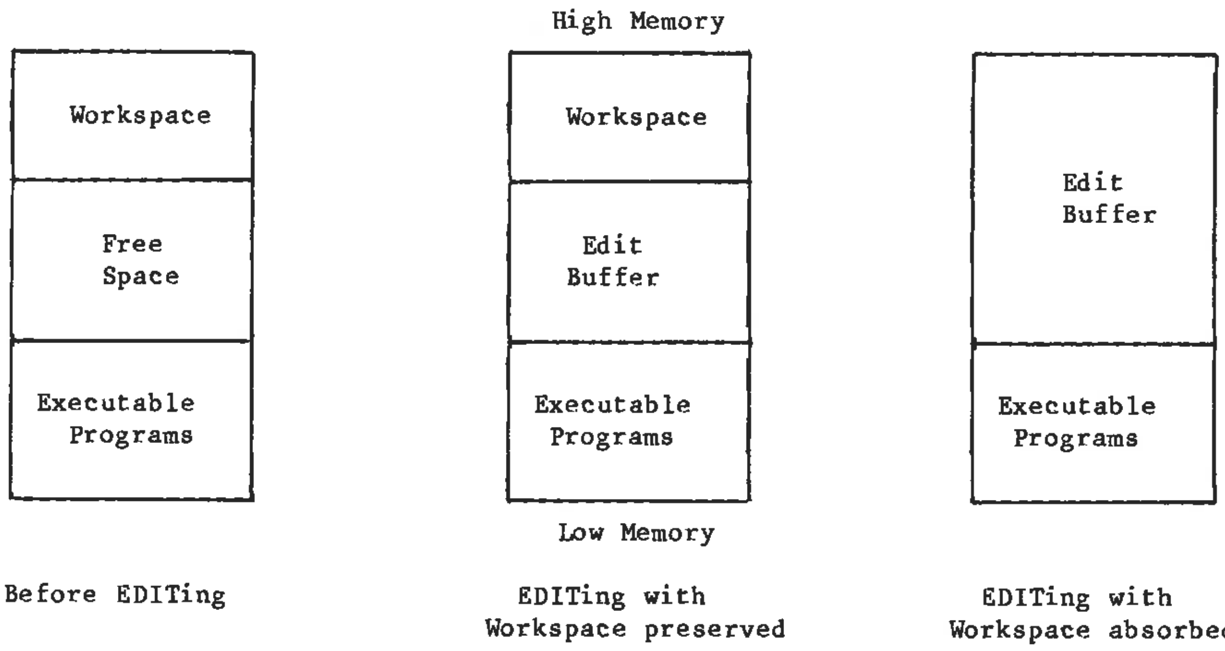
As a final note, the PROMAL EDITOR is itself written in PROMAL. We think that you will find this fact significant if you have ever considered writing your own editor, word processor, or similar program in a high-level language. Obviously, PROMAL can do the job!

TECHNICAL NOTES ON THE EDITOR FOR THE COMMODORE 64

The remainder of the section on the EDITOR applies only to the **Commodore 64.**

RELATIONSHIP OF THE WORKSPACE TO THE EDIT BUFFER

The EDITOR uses all unallocated memory as a buffer for editing your text. The size of unallocated memory can be displayed by the EXECUTIVE MAP command. Depending on how you enter the EDITOR and whether the Workspace is empty or not, the Workspace (W) may or may not be "absorbed" into the edit buffer:



The Workspace will be "absorbed" into the Edit buffer (to maximize your available editing buffer size) if:

1. The Workspace is empty, or,

2. The Workspace is not empty and NO file name was specified on the EDIT command (in this case the Workspace is absorbed non-destructively).

The Workspace will NOT be absorbed into the Edit buffer if:

1. The workspace is NOT empty and a file name or W was specified on the EDIT command.

If the Workspace is absorbed, you cannot WRITE or RECALL from the Workspace during the EDIT session. If the session was started with an EDIT command without a file name or W specified, the Workspace will be automatically updated when you quit the EDITOR. If the Workspace is not absorbed, you may use it like a file during the EDIT session (for SAVEs and RECALLS).

In any event, the Workspace will be "un-absorbed" when you exit from the EDITOR back to the EXECUTIVE. The EXECUTIVE WS command can be used to clear the Workspace or alter its size before entering the EDITOR.

EDITOR COMMODORE 64 CHARACTER SETS

The EDITOR creates and updates files of standard ASCII characters. Unfortunately, the Commodore character-generator ROM's do not conform completely to the ASCII standard. As you may know, there are two sets of characters which can be displayed on the screen, upper case plus graphics and upper plus lower case. The PROMAL EXECUTIVE selects upper and lower case mode, and this is the mode that should normally be used with PROMAL. The ALPHA LOCK (CTRL A) feature can be used to automatically generate upper case letters in this mode, if you so choose.

There may be times, however, when you might want to edit in the upper case and graphics mode. You can switch modes by pressing SHIFT C= (shifted Commodore key). When the EDITOR starts up, it senses the currently selected mode and edits accordingly. Also, when the EDIT HELP menu is displayed, you may switch modes and then resume editing with the C selection. Selecting the upper case and graphics mode will permit you to enter the various non-ASCII characters supported by Commodore. For installing special characters into strings in a PROMAL program (such as to change colors or reverse video), we recommend that you use the special hex literal form described in the PROMAL LANGUAGE MANUAL instead of using graphics keys (for example, "£12 Hello! £92" will generate a reverse video string, " Hello! "). When you exit back to the PROMAL EXECUTIVE, whatever character set you entered the EDITOR with will be restored.

THE PROMAL COMPILER

The heart of the PROMAL system is the PROMAL compiler. This program takes as input a file containing your source PROMAL program (which you normally generate with the editor). It produces as output a small, very-fast executing command file (object program). This object program can be executed from the PROMAL EXECUTIVE by merely typing its name. Optionally, the COMPILER can also produce a listing file. The listing shows your original source program, with line numbers and the addresses assigned to variables and statements.

Unlike BASIC, you cannot execute a program in the form you type it in. First it **must** be compiled. This is a great benefit, since the compiled program will only be a small fraction of the original size, and will execute many times faster than it would be possible to execute the original text version of the program with an interpreter, as is done in BASIC. The compiler does not alter your source file; it is left intact so you can examine it or edit it further. Anytime you make a change to the source program, you will need to re-compile it before the changes will take effect in the object program.

Unlike the EDITOR and the EXECUTIVE, the COMPILER is not automatically loaded into memory when PROMAL boots up. This is because the COMPILER is a large program (about 15K bytes), and it is only needed when you want to compile a new program (or revised old program) into executable form.

STARTING THE COMPILER

The COMPILER can be run from the EXECUTIVE with a command of the form:

COMPILE [Sourcefile][**O**=Objectfile]][**L**[=Listfile]][**V**=c][**B**[=Solf]]

The simplest form of this command is just:

COMPILE

which will compile a source program from the Workspace and produce an executable object program on disk, ready to run by typing its name. The object program written will have the same file name as the name on the PROGRAM line of the source program, with a ".C" extension.

If you compile a program which already has an existing object file on the disk, the compiler will prompt:

```
REPLACE EXISTING Filename.C (Y/N)? _
```

Press Y and RETURN to replace the old file; any other reply will terminate the compilation. After a successful compilation, the EXECUTIVE will automatically unload any existing copy of the compiled program from memory. This ensures that you will always execute the new version of a freshly compiled program, not an old version which might still be in memory.

For the **Commodore 64:** The demo compiler can be used to compile small programs directly from the Workspace. **When using the full compiler (which is larger than the demo compiler), you will want to keep the Workspace clear and compile directly from disk.**

For the **Apple II:** The compiler is always loaded from disk when needed, and unloads when it is finished, so you will have more room available for your programs. It is a good idea to give an UNLOAD command before compiling.

Most frequently you will compile directly from a source program on disk and produce an object file on disk. For example:

```
COMPILE MYPROG
```

will tell the compiler to read the file MYPROG.S (the compiler supplies the ".S" default extension) and produces an object file called MYPROG.C. The COMPILER gives the object file on disk the same name as the source file, but with a ".C" extension.

You can use the "O" (the letter "oh") option to output the object code to a file other than the same name as the source file. For example:

```
COMPILE MYPROG O=NEWVERSION
```

will read MYPROG.S and generate NEWVERSION.C. (Note: we strongly recommend that you **use the same file name for your object file as is declared on the PROGRAM header line of the source program.** Otherwise, the EXECUTIVE will load the FILE you specify from disk each time you type its name even if the same program is already in memory, because it does not match the command name. The name on the PROGRAM statement is the name which appears in the MAP display.)

If you add the "L" option to your command, a listing will be produced. The "L" option by itself will cause the listing to be written to a file with the same name but with a ".L" extension. For example:

```
COMPILE MYPROG L
```

will read MYPROG.S, produce an object file MYPROG.C, and listing file MYPROG.L. If the listing already exists it will be replaced. You can also send the listing directly to the printer by specifying **L=P.**

The V option is used to specify a version of your program for conditional compilation, described in Chapter 5 of the PROMAL LANGUAGE MANUAL.

The final option for the COMPILE command is the "B" option, which is used to compile big programs from disk. This option will not normally be needed unless you are compiling fairly large programs. The B option is not available on the Demo compiler. It is easy to tell when you need to specify the B option. The statistics displayed at the end of a compilation tell you how much of the COMPILER's available table space was used. If any of the percentages shown start approaching about 90 percent, you probably should be using the B option. When compiling large programs, <u>you should be sure to remember to UNLOAD</u> other programs before compiling, because the COMPILER uses all available memory for its tables. For example:

```
UNLOAD
COMPILE MYPROG B
```

will read MYPROG.S and produce an object code file MYPROG.C, allowing <u>very</u> large programs to be compiled (typically several thousand lines).

The B option changes the way the compiler uses memory, so it will have more memory for its tables. Normally the compiler stores the compiled object code in a buffer in memory until the whole program is compiled, and then writes it to disk. When the B option is selected, it writes the object code to disk "on the fly" instead. For the **COMMODORE 64** only, the "B" option will also delete the EDITOR from memory, making enough room available to compile very large programs. The EDITOR will be reloaded from disk when you need it.

For **very** large programs it is possible that one of the tables used by the compiler may overflow even with the B option. For example, if your program uses many hundreds of lines of strings in data statements, the string table may get full (as indicated by the statistics displayed at the end of compilation). In this case, you can tell the compiler to allocate more of available memory for the particular table which is overflowing, using this form of the B option:

```
COMPILE Myprog B=Solf
```

where **Solf** represents a four digit hexadecimal number, where the four individual hex digits must sum to exactly 10 hexadecimal. For example:

```
COMPILE MYPROG B=C112
```

Each digit represents what part of available memory is to be allocated for a particular table, in units of 1/16th of available memory, as follows:

```
S (first digit) = number of 16ths used for symbol (name) table
O (second digit) = number of 16ths used for object buffer
L (third digit) = number of 16ths used for literals (strings)
F (fourth digit) = number of 16ths used for forward references
```

Therefore the example above used 12/16 ($C=12 decimal) of available memory for the symbol table, 1/16th for the object code buffer, 1/16th for strings, and 2/16ths for forward references. The default for the B option is B=A132. The value of the second (O) digit should always be left at 1.

For the **COMMODORE 64**, you should never attempt to use the L option to produce a listing file on disk at the same time as using the B option. This is because the 1541 disk drive will only allow one file to be open for writing at a time while a file is open for reading, and the B and L options each require one output file. If you want to get a listing, use L=P or make a second compilation with O=N to suppress the object file.

DIALOGUE OF THE COMPILATION PROCESS

When the COMPILER starts, it will run continuously until your program is compiled or until an error is detected, without further action on your part. After it "signs on", it will immediately read in your your source program, and show a rapidly changing message:

```
READING LINE nnn
```

where nnn is the line number the compiler is reading. This number will change rapidly (several lines per second), especially if you are compiling from the Workspace. When the compiler encounters an INCLUDE LIBRARY statement in your program, it will display another line saying:

```
    INCLUDING LIBRARY
```

and will then generate another line of the form

```
    READING LINE nnn
```

as it scans the LIBRARY. This takes about 5 seconds. It will then display:

```
    RESUMING FILE 0:Filename.S
```

indicating that it has finished the INCLUDE file and is resuming reading the original file (the "0:" indicates the disk drive number for Commodore 64 versions; for Apple II this will be replaced with the path name). Also, each time the COMPILER reads a new Procedure or Function definition in your program, it displays a message showing the start of the subroutine.

If an error is detected in your program, the COMPILER will halt and display the offending line, followed by an error message. For example:

```
    READING LINE 303
    COMPILING FUNC BYTE READJOYSTICK
    READING LINE 309
           IF FIREBUTTON OR UP OR DOWN
    *********************

    ERROR 27:
    UNDEFINED

    Want to EDIT W (Y/N)? _
```

This indicates that the variable FIREBUTTON has not been defined (declared). The row of asterisks indicates how much of the line the COMPILER had read before the error was detected.

The line "Want to EDIT W (Y/N)?" asks if you want to edit the file containing the error (in this case, the Workspace, W). If you reply with Y (and <RETURN>), the EDITOR will start and automatically position the cursor to the offending line, simplifying the correction process. Otherwise, you will be returned to the EXECUTIVE.

When looking at a compiler error message, note that the row of asterisks may not always indicate the **exact** location of the problem. The asterisks only indicate how far the compiler had read before it recognized an error. The error could have been caused by anything up to this point in the program. For example, a misspelled variable declaration near the beginning of the program will not cause an error until referenced with the "correct" spelling later in the program. **Appendix C** contains a listing of all Compiler error messages with an explanation and corrective action for each. The compiler error messages are contained in file COMPERRMSG.T. If this file is not present on your disk, you will only get an error number, without the text of the message.

Assuming your program compiled to completion without an error, you will see a signoff message with a summary. This indicates that your program compiled successfully and is ready to execute. You are back in the EXECUTIVE. The summary shows the total number of lines read by the compiler, the number of bytes of object code generated, the number of bytes of memory your program will need for variables, and how much of the compiler's tables were used. Numbers with a $ prefix are hexadecimal, and those in parentheses are in decimal. Note that the actual object file will be somewhat larger than the number of bytes of object code generated, because the object file contains additional header and relocation tables.

DIFFERENCES BETWEEN VARIOUS PROMAL COMPILERS

The Demo compiler and standard compiler are both named COMPILE. The Demo compiler is found on the demo disk and the full compiler on the PROMAL system disk. You may prefer the Demo compiler for small program development.

The Demo compiler does not support the B option (described above), nor does it support IMPORTs, EXPORTs, or separate compilation (described in Chapter 8 of the PROMAL LANGUAGE MANUAL), and is limited to a maximum of 400 source lines (excluding comments but including the LIBRARY).

For the **Apple II,** if you use the EXECUTIVE command GET COMPILE, the standard compiler will not show up in the memory MAP and cannot be executed from memory, because there is not enough room in memory for the compiler and the EXECUTIVE at the same time. This is of no consequence, except that you need to keep a copy of the compiler on each of your disks you use for program development.

PROMAL CROSS REFERENCE MAP UTILITY

XREF generates an alphabetized list of all the identifiers (names) used in a program, with a sorted list of line numbers on which they appear. This list is very useful for locating where variables, procedures, or functions are used in a program. The XREF utility is normally used after a compilation with a listing generated. In this case, XREF will append the cross-reference map to the end of the listing file. The syntax of XREF is:

```
XREF [Sourcefile][V=c]
```

where Sourcefile is the name of the PROMAL sourcefile to be read. It may be good idea to UNLOAD any other programs before running XREF, so it will have plenty of room for the tables it uses internally. The optional second argument **V** specifies a version for conditional compilation. If you are using conditional compilation, you should specify the same V option that you use on the corresponding COMPILE. Omitting the V will cross reference all conditional blocks.

When XREF is run, it appends the cross reference map to the end of the listing file, if one exists. Otherwise, it generates a listing with line numbers (but does not generate any code or do any syntax checking like the compiler) and a cross reference map, on a file with the same name as the source file but with a **.X** extension. This is useful if you wish to obtain a cross reference map for a program which will not successfully compile yet.