P R O M A L

(PROGRAMMERS MICRO APPLICATION LANGUAGE)

DEVELOPER'S GUIDE

For Apple II and Commodore 64 Computers

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

Rev. C - Sep. 1986

TABLE OF CONTENTS

INTRODUCTION.....3

GENERAL DESCRIPTION.....3

SPECIAL CONSIDERATIONS.....4

MEMORY MAP DIFFERENCES.....5

WORKSPACE (W DEVICE) CONSIDERATIONS.....7

SIZING YOUR APPLICATION PROGRAM.....8

EXIT, ABORT, AND RUNTIME ERROR PROCESSING.....9

SPECIFYING YOUR OWN ERROR RECOVERY.....11

LOADING MACHINE LANGUAGE ROUTINES.....11

BUILDING YOUR MASTER DISKETTE.....12

COPY PROTECTION.....12

MISCELLANEOUS.....12

FINAL NOTES.....13

TABLES & FIGURES

FIGURE 1: Memory Maps.....5

TABLE 1: Runtime Error Codes.....10

PROMAL DEVELOPER'S GUIDE

INTRODUCTION

With the PROMAL DEVELOPER'S SYSTEM and this manual, you will be able to generate diskettes containing your compiled PROMAL programs, which can be run on computers which **do not** have the PROMAL system. This will allow you to sell or distribute your application programs written in PROMAL to users who do not own PROMAL. As long as the acknowledgement requirements of the License Agreement are met, no royalties or other payments will be due to SMA, regardless of how many copies you sell or distribute.

If you have been using the PROMAL system, you already know that it is far superior to BASIC for developing programs, and that programs execute much faster. As a program author, you will also appreciate a very significant advantage of PROMAL: since you only need to distribute the compiled object code, you have a greater degree of security from unauthorized copies or modifications. You do not have to provide the source code.

By using the DEVELOPER'S SYSTEM, you will be able to make disks which "boot up" similar to the PROMAL Demo diskette, except that instead of running the PROMAL EXECUTIVE and EDITOR, your application program will be executed. You have control over what the programs are named and what program will be run. Your application can also load any other PROMAL or machine-language programs it might need.

GENERAL DESCRIPTION

The rest of this manual assumes that you already have a basic working familiarity with the PROMAL system.

When you "boot up" the regular PROMAL system, you LOAD a file called "PROMAL" and RUN it on the Commodore 64, or autoboot PROMAL.SYSTEM on the Apple II. This file contains the PROMAL nucleus and all the routines in the library. This file is called the runtime package. When this PROMAL runtime package is run, it automatically loads the EDITOR and EXECUTIVE, as well as the LIBRARY definitions, and begins execution of the EXECUTIVE.

The PROMAL DEVELOPER'S SYSTEM contains a special version of the runtime package which loads and runs your application program instead of the EXECUTIVE and EDITOR. You copy this runtime system and your application program onto a **MASTER DISKETTE** using a special program called **GENMASTER**. You may then duplicate and distribute your master diskette.

For the Apple II, your application will autoload on power-up just like PROMAL. For the Commodore 64, a user will LOAD the runtime package (which you can name anything you want) from the master diskette and RUN it. The runtime package will autoload your application program and execute it. No PROMAL signon message or other indication that this is a PROMAL program will be displayed; your program will be completely in control. Since there is no EXECUTIVE or EDITOR, when your program terminates (if it does), it will exit as described for procedure PROQUIT in the PROMAL Language Manual.

There is presently no provision for generating Commodore 64 cartridges or ROM-resident code.

The general procedure you will use for preparing an application program for distribution is:

1. Write, compile and test your application program using the regular PROMAL system.

2. When you are satisfied that your program is correct, use the GENMASTER program provided with the Developer's disk to bind your compiled application program with the special runtime package, and copy it to your desired master diskette.

3. Duplicate the master diskette and distribute it as you see fit. **You must include the acknowledgement notice: "(program) is a licensed PROMAL application program. PROMAL is a trademark of Systems Management Associates, Inc., Raleigh, NC.".** This notice must appear on the diskette label, in the documentation, or on an initial screen display of your program.

This manual only addresses Step 2. But before discussing the mechanics of Step 2, we will examine some of the special considerations which you should be aware of when preparing an application program for mastering.

SPECIAL CONSIDERATIONS

The main difference between running your application under the regular PROMAL system and as a dedicated application is that you will no longer have the EXECUTIVE available. Therefore:

1. The user will not be able to issue EXECUTIVE commands.

2. Your program cannot receive command arguments from the EXECUTIVE.

3. Your program cannot EXIT or ABORT to the EXECUTIVE (if you do EXIT or ABORT, the PROQUIT procedure will be executed).

4. If your program encounters a fatal programming error which would normally abort back to the EXECUTIVE (such as division by zero or calling a library routine with an illegal number of arguments), it will not return to the EXECUTIVE. A method is provided for you to recover from these errors within your application program (discussed below).

5. Since you have no EXECUTIVE to execute selected programs in memory, only your one application will be in memory at one time (unless you LOAD additional programs from within your program).

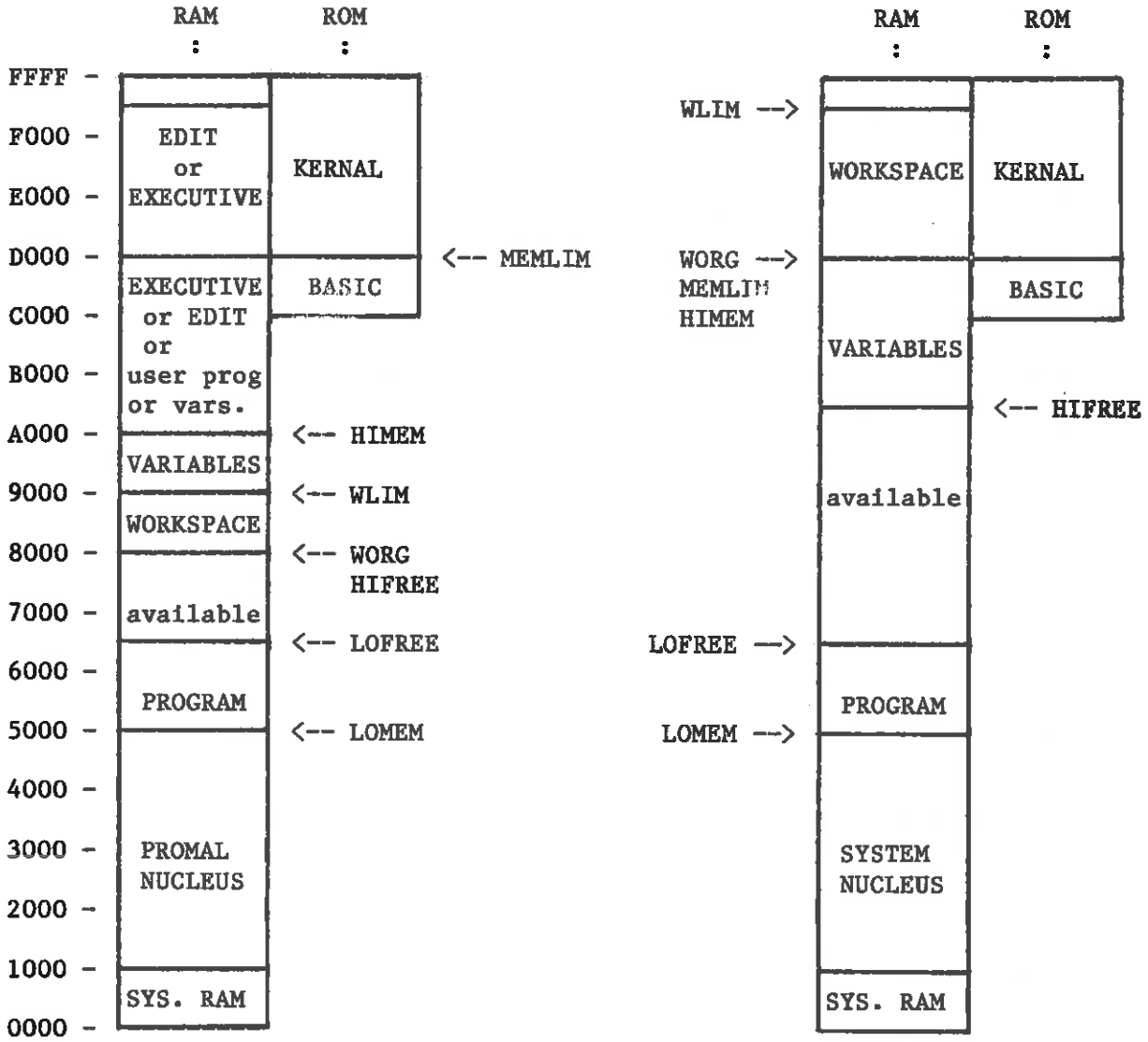
6. The EXECUTIVE automatically closes any open files when you exit a program in the normal PROMAL environment. Since you have no EXECUTIVE, you should be sure to close any files that you opened.

MEMORY MAP DIFFERENCES

Figure 1 shows the difference between the memory map for a typical compiled PROMAL program when loaded by the standard PROMAL system (left), and by the developer's master disk (right). The primary difference is that the master disk version does not have the EXECUTIVE or EDITOR in memory.

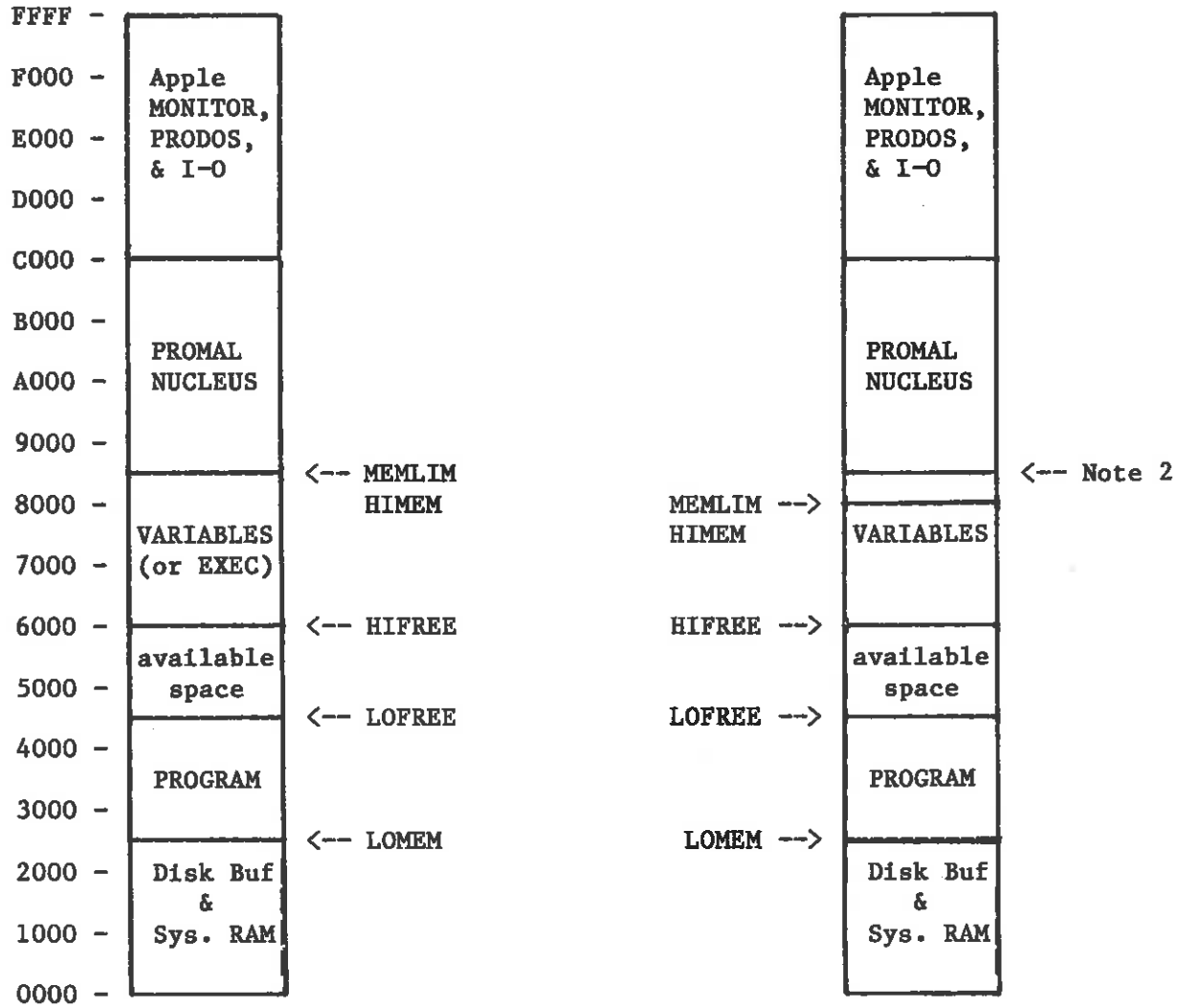
Figure 1

Memory Map - Commodore 64



Note: All addresses shown are approximate. See Appendix G of the PROMAL Language manual for locations of the memory pointers. $FE00-FFFF holds the Function key definition buffer & CTRL-B buffer (both versions).

Memory Map - Apple II



STANDARD PROMAL DEVELOPER'S APPLICATION

Notes:

1. All addresses shown are approximate. See Appendix G of the PROMAL Language Manual for locations of the memory pointers.

2. In the Developer's version, MEMLIM is 1 page lower than in the normal version, to allow room for the Function key definitions (which are normally in banked memory). The CTRL-B buffer is in main memory too, but is included in the Nucleus for the Developer's version.

The PROMAL NUCLEUS and SYSTEM NUCLEUS are very similar, except that the SYSTEM autoloads your application program instead of the EXECUTIVE and EDITOR when it is started up (NOTE: The developer's SYSTEM file **cannot** be used to run the EXECUTIVE or EDITOR). The application PROGRAM is identical in both cases, and is simply your compiled PROMAL program.

Your application program will load in the same location as it would with the normal PROMAL system with no other programs in memory. Your variables will be loaded in the same way as normal, at the top of available memory (by default), or immediately above your program code (if you specified OWN on the PROGRAM line).

The only difference will be that on the Apple, you will have one page (256 bytes) less space at the top of free memory. This is because the Apple developer's nucleus supports the function key buffer and backtrack (CTRL-B) buffer in main memory instead of in bank-switched memory. This permits your PROMAL application to run on Apple IIe systems **without** the extra bank-switched memory. Your programs will also run on IIe computers without 80 column boards. In this case, your programs will run in 40 column mode. If you wish, your program can test for the presence of an 80 column board as described in the Apple reference manuals. Your application should not use the Workspace (W device) on Apple computers without the extra bank-switched memory present.

If your Apple II program does not use function substitution strings, you can use the page of space for a buffer, etc. To do this, move MEMLIM up one page, set the key codes for BKEYFKl and BKEYFKL to $00 (see Appendix G), and use the page immediately below MEMLIM as you see fit.

WORKSPACE (W DEVICE) CONSIDERATIONS (COMMODORE 64 ONLY)

For the Commodore 64, another difference in the memory map is the location and size of the Workspace. When your application runs from your master diskette, the Workspace will be initialized to point to a large (11.5K byte) area starting at $D000. This will have no effect on your program unless you use the workspace from your program (the W device). This space is in the RAM of the Commodore-64 which is "under" the ROM. Reading or writing the W device automatically performs the necessary "bank-switching" to access this RAM. If you wish, you may move the location of the workspace from the application program by manipulating the pointers before starting to use the W device.

The file PROSYS.S defines five global variables associated with the Workspace:

```
WORD WORG  ($0DC5) points to the start of the available workspace.
WORD WPTR  ($0DC7) points to the next byte to be read or written.
WORD WEOF  ($0DC9) points to end-of-file (byte after the last byte written).
WORD WEND  ($0DCB) points to the byte after the last byte of the workspace.
WORD WSIZE ($0DDB) is the size of the available workspace (=WEND-WORG).
```

WORG and WEND will not change unless you change them (in the Developer's System version). This differs from the standard PROMAL system environment, where the workspace moves dynamically as programs are loaded and unloaded, and is also controlled by the WS command of the EXECUTIVE. On either system, WPTR and WEOF are moved automatically by the W device driver. A workspace with nothing in it has WEOF = WPTR = WORG.

Suppose you wish to increase the Commodore Workspace size to use all available memory. Before doing any input/output to the W device, you could execute these statements:

```
INCLUDE PROSYS
...
WORG = LOFREE    ; move start of W down to start of free memory
WPTR = WORG      ; set workspace to empty
WEOF = WORG
HIFREE = LOFREE ; no more free space now
WSIZE = WEND-WORG
...
```

Besides using the available memory for an increased workspace, you could also use it for loading machine language routines, for dynamically-allocated buffers, for hi-res screen memory, or for whatever you need. The available space starts at LOFREE and goes up to HIFREE, just as it does in the standard system; the available space is just bigger.

WORKSPACE CONSIDERATIONS (APPLE II)

The Apple II Developer's version keeps the same Workspace as the standard system.

The Workspace in the Apple is in bank-switched memory. Therefore if your application program is intended to run in IIe computers without any extra memory (40 column mode), you may not use the W device.

SIZING YOUR APPLICATION PROGRAM

For the Apple II:

Your new Master disk will have on it a copy of the ProDOS system, the PROMAL runtime system, and your compiled PROMAL application program. You should copy the PRODOS file onto the master disk after you format it using the ProDOS Utility disk. The remaining files will be copied onto the master disk when you run the GENMASTER program.

Your application program will have virtually the same amount of space and will run at the same location as it does under the standard PROMAL system. When GENMASTER runs, it will ask you if you wish to include support for REAL arithmetic in the runtime package. If you say no, this is equivalent to executing a NOREAL command from the EXECUTIVE, and will reduce the size of the runtime package by about 2.5K bytes and increase the available space for your application by the same amount. You will also be asked if you wish to reserve space for the hi-res graphics page. Answering yes is equivalent to executing a BUFFERS HIRES command from the EXECUTIVE. Another prompt will allow you to specify the number of disk buffers, which is equivalent to a BUFFERS command in the EXECUTIVE.

The amount of disk space needed will be the sum of the PRODOS file (about 15K bytes), the RUNTIME.SYSTEM file (about 19K bytes), and your compiled PROMAL application program. You may add any additional files you need to the master disk after running GENMASTER.

For the Commodore 64:

The GENMASTER program on the DEVELOPER'S DISK will copy the special runtime package plus your compiled application program onto your new Master diskette. Actually there are two runtime packages provided, one with and one without REAL support. The GENMASTER program will ask you which you wish to use. The standard SYSTEM file is approximately 18K bytes, about the same as the PROMAL file on the standard PROMAL system. Therefore your program will be loaded into memory at the same address as the start of allocatable memory in the standard version (as shown by the MAP command).

If you select the SYSTEM without REAL support, the SYSTEM file will be reduced by about 2.5K, which will reduce the load time and enable your application to be loaded at the same address as in the standard system after executing a NOREAL command.

GENMASTER will unload the EDITor from memory while it copies the system nucleus to your master disk. It will be automatically reloaded from disk later if you need it. You may not use the EDITor, EXECUTIVE, or COMPILER as an application program for GENMASTER.

The amount of disk space used on your Master disk will simply be the sum of whichever SYSTEM file you select plus the size of your compiled application program. You may copy any additional files you need using the EXECUTIVE COPY command.

EXIT, ABORT, AND RUNTIME ERROR PROCESSING

In the normal PROMAL system, when your program executes to completion (or executes a call to the EXIT or ABORT library routines), control is returned to the EXECUTIVE. When you exit from your program on a Master disk generated by the Developer's System, you don't have the EXECUTIVE to go back to, so the PROQUIT procedure will be called instead. In a dedicated application program, you may not want to have any exit at all.

This brings us to the subject of error recovery. There are several different kinds of runtime errors that can occur during execution of a PROMAL program. First, there are normal I-O errors (such as attempting to open a disk which is not in the drive), which return error indications to your program so that you may take whatever corrective action is required under program control. These kinds of errors work precisely the same under either the standard system or on your master disk system.

However, there is another class of more serious errors, the kind that "shouldn't happen" in a debugged program. In the standard system, these kinds of errors abort back to the EXECUTIVE with an error message. For example, if you divide by zero, you will get a message that says:

```
*** RUNTIME ERROR:  0-DIVIDE
AT $47BA
*** PROGRAM ABORTED.

-->
```

When you run a program from your Master Diskette, you do not have an EXECUTIVE to abort to. Instead, you can handle these kinds of errors in two ways: (1) do nothing, in which case a default error message will be printed and the computer reset; (2) provide your own error recovery using a REFUGE 3, as described below. If you do nothing and a fatal error occurs, the system will display an error message similar to this:

```
FATAL SYSTEM ERROR $0D AT $47BA
PRESS ANY KEY TO RESET COMPUTER
```

When the user presses a key, the program will exit via PROQUIT. Again, remember that this only applies to the kinds of errors that would abort a program, which should not be present in a debugged application. The meanings of the error code displayed are given in Table 1.

Table 1
Runtime Error Codes

Error #	Meaning
1	Machine language breakpoint ($00) encountered. (Note: the address of the breakpoint and register contents can be found in these locations: address - $0C51; A - $0C53; X - $0C54; Y - $0C55)
2	PROMAL breakpoint ($00) encountered at the indicated address. Usually caused by a corrupted program.
3	(reserved)
4	Stack overflow. Generally caused by too many levels of nested subroutine calls in combination with large numbers of local variables.
5	Illegal opcode. Usually caused by a corrupted program (array out of bounds, bad pointer, block move error, etc.)
6	Divide by 0 (real, integer, MOD or real overflow).
7	Required software package not loaded (for example, floating point arithmetic without REAL support loaded).
8	Illegal number of arguments on FUNC or PROC. A Library routine was called with too many or too few arguments.
9	I-O direction error. For example, trying to open the printer for read access.
A	Illegal argument for FUNC or PROC. The argument for a library routine is out of range.

B Illegal file handle. Tried to perform I-O to a file or device that was not properly opened, or missing or defective file handle argument.

C I-O redirection error. Tried to redirect to an unopened or illegal file or device.

D CTRL-STOP key pressed (Commodore 64) or CTRL-RESET (Apple II).

SPECIFYING YOUR OWN ERROR RECOVERY

If you wish, you may provide your own error recovery to recapture control within your application program after these errors. To do so, define a REFUGE 3 in your program, and control will return to this point on a fatal error. You can determine what the error was by inspection of BYTE DRTERR AT $ODF6, which will contain the error code as listed above. CAUTION: You should not add this error recovery code to your program until your application is otherwise completely debugged, as it will interfere with normal PROMAL error recovery to the EXECUTIVE. An example of error recovery is shown in the program fragment below.

```
BYTE FATAL_ERROR          ; Set false while defining REFUGE, then TRUE
EXT BYTE DRTERR AT $ODF6 ; system runtime error #
...

; Define runtime error recovery entry point...
FATAL_ERROR = FALSE       ; Don't recover while just defining refuge
; Come here on fatal runtime error only..
REFUGE 3
IF FATAL_ERROR
  OUTPUT "#CUnexpected system error $#h",DRTERR
  PUT CR,"Attempting to close all files..."
  ... ; (close files, cleanup whatever you can here)
  PUT CR,"Restarting computer now..."
  ABORT
FATAL_ERROR = TRUE        ; Turn on error recovery now
...
```

The purpose of the FATAL_ERROR variable in the fragment above is to avoid executing the recovery code itself while you are merely trying to define its location by executing the REFUGE 3 statement. Don't forget that in order for a REFUGE to be active, it must be **executed**, not just exist somewhere in your program.

LOADING MODULES AND MACHINE LANGUAGE ROUTINES

Since you cannot use the EXECUTIVE GET command to load separately compiled modules or machine language routines, your application will have to load these programs itself (if it needs any). You should use a bootstrap loader as described in Chapter 8 of the Language Manual to perform this function. This also applies to programs in the optional Graphics Toolbox. For machine language routines, you can use either the LOADer or function MLGET, as described in Appendix I.

BUILDING YOUR MASTER DISKETTE

A special PROMAL program called GENMASTER is provided on the DEVELOPER'S DISKETTE. This is an interactive program which is used to copy the PROMAL runtime nucleus and your compiled PROMAL application program onto the disk you want to become your Master Diskette (which you should have pre-formatted). For the Apple version, you should also copy the PRODOS file onto your newly formatted master disk. You should run the GENMASTER program from the regular PROMAL system. To run the program, insert the DEVELOPER'S SYSTEM DISKETTE into the drive, and type:

For the Apple II:

```
UNLOAD
PREFIX *
GENMASTER
```

For the Commodore 64:

```
UNLOAD
GENMASTER
```

This program is self-explanatory through its prompts. Simply pressing RETURN will select the default. RETURN can also be used as a "YES" response to prompts. You will have the opportunity to specify the name of the runtime system and the program on the new master disk. For the Apple II, the runtime system name must end in ".SYSTEM" to be acceptable to ProDOS. Once your Master disk is made, it can be booted up like any other program. For the Apple II, the program will auto-boot when the computer is powered on or CTRL-APPLE-RESET is pressed. For the Commodore 64, you will type:

```
LOAD "SYSTEM",8
RUN
```

This of course assumes you accepted the default name of "SYSTEM".

COPY PROTECTION

If you are developing a commercial program, you will want to decide whether or not you wish to employ some scheme to protect your disk from illegal copying. This is entirely up to you. PROMAL neither encourages nor discourages the use of copy protection schemes. Generally any protection scheme which you can use with BASIC or machine language can also be employed with PROMAL, if you wish. If you wish to employ copy protection, consult your mass disk duplicating service or books on the subject for available techniques. A discussion of copy protection techniques or ethics is beyond the scope of this manual.

MISCELLANEOUS

1. For the Commodore 64, DYNODISK works exactly the same as for the regular system. If you simply want DYNODISK disabled in your application, you can apply the same patch to the SYSTEM or SYSTEM_NR file as is described for file PROMAL in Appendix N.

2. For the Apple II, the /RAM device will be handled exactly as in the regular system. Therefore it will be enabled if the users has more than 128K and has installed the /RAM device. Your application can be booted from /RAM in the same way as regular PROMAL.

3. If you have purchased the optional source code to PROMAL, you **may not** include it in whole or in part in any application you sell or distribute, nor can you include the Compiler (not the Demo Compiler either).

4. If you use multiple modules (as described in Chapter 8 of the PROMAL Language Manual) and have an ESCAPE in one module to a REFUGE in a separate module, if you exit from the module with the ESCAPE via a normal END, the program will still return to the parent program of the original module.

5. For IBM PC PROMAL there is only a Developer's version (no End-User's version), because programs are run under DOS.

6. A Source code listing on disk for the Developer's Runtime package (as well as the regular runtime package and library) is available as an option from SMA. Please call for ordering information.

7. For the Apple, you may wish to have your program ignore CTRL-C instead of aborting. This can be done by setting the byte at $0DE0 to $00.

FINAL NOTES

Check your DEVELOPER'S DISKETTE to see if a file called README.T is present on the disk (using the FILES command). If so, type this file before attempting to build your first Master diskette. If present, this file contains additional information not covered in this manual. Other additional files may also be provided on the Developer's disk for informational purposes.

This page is intentionally left blank

