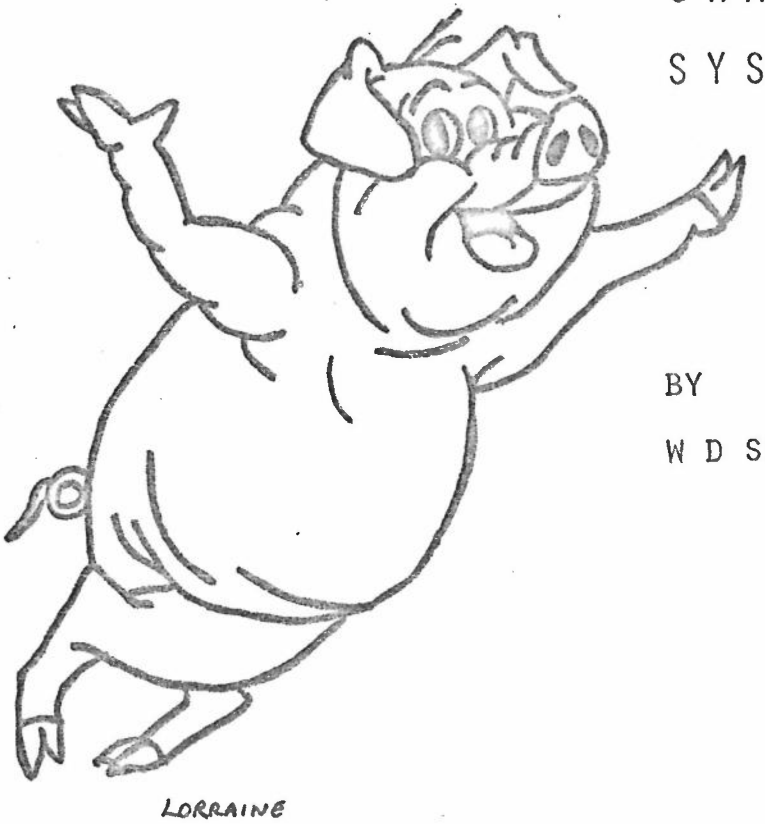
ATLAS COMPUTER LABORATORY

PIGS MANUAL

SCIENCE RESEARCH COUNCIL

PIGS

PDP15 INTERACTIVE GRAPHICS SYSTEM



BY

W D SHAW

Atlas Computer Laboratory
Chilton
Didcot
Oxfordshire
OX11 0QY

October 1974

CONTENTS

INDEX OF FIGURES AND TABLES

INTRODUCTION

INTRODUCTION

PIGS is a set of procedures which simplifies the design of interactive graphics programs on the PDP15. The system is composed of an interpretive language, MENDEL (for MENU DEFINITION LANGUAGE), and a collection of procedures called the PIGS Run Time Environment (PRTE).

In designing a Graphics Application Program (abbreviated GAP), MENDEL is used to describe the organisation of commands available to the operator into groups called menus. PRTE uses the binary output of the MENDEL assembler to display menus on the CRT, thus suggesting contexts of available commands. When an operator selects a command and defines its arguments, control is passed to a pre-specified user procedure to perform the desired function. It is possible for a GAP to share many of the PRTE subroutines to save core and provide more flexible interaction.

This manual explains how to operate and design graphics applications programs using PIGS. Chapter 1 contains a description of facilities available to an operator, while the remaining chapters provide information of use primarily to the GAP designer. Some knowledge of the DOS operating system, overlay system(XCHAIN) and graphic code-generating package (FOG) is assumed. The recommended documentation covering these topics is:

(1) PDP15 User Note 1
(2) DOS Users' Manual
(3) CHAIN and EXECUTE Manual, PDP15 User Notes 3, 4
(4) FOG - FORTRAN GRAPHICS on the PDP15

Throughout the manual examples will be taken from a demonstration program, PATH, which is described in Appendix A according to the documentation suggestions given in Chapter 7.

PATH is available on DECtape 155 for practice and experimentation.

1. OPERATING PIGS

This chapter contains operational information common to all graphics application programs using the PIGS run time environment. In order to completely understand a particular GAP, it will also be necessary to consult the designer's documentation of its commands, displays, and data base which is available in the Graphics Application Program Library.

Examples in this and subsequent sections are drawn from a simple interactive graphics program called PATH. This graphics application has commands which allow an operator to draw an object and the path it is to follow using a computer-read stylus. The operator may command the program to play back the animation on the display at various speeds. It might be a good idea at this point to read the general description of PATH and its commands given in Appendix A, parts (4) and (6).

1.1 Loading

Assuming that a GAP has been debugged and is ready for use, it will normally be available on a DECtape in the racks near the PDP15. The number of the tape is given in the program documentation.

Since the PIGS run time system is very large, GAPs will always have been organized into overlays using the systems program, XCHAIN. XCHAIN outputs files with several different extensions which must all be on the PDP15 disc at run time. For any given application there will be two binary files on the tape with extensions XCT, and XCU, and a set of *n* binary files with extensions LØ1, LØ2,...LØ*n*. In addition, the MENDEL assembler passes menu and command information to PRTE via a binary file with extension MNB; this file must also be transferred to the disc. As an example, PATH consists of the following 8 binary files on DECtape 155.

```
PATH XCT
PATH XCU

PATH LØ1
PATH LØ2        From XCHAIN
PATH LØ3
PATH LØ4
PATH LØ5


PATH MNB        From MENDEL
```

To move these files from tape to disc it is necessary to use the DOS operating system and the Peripheral Interchange Program, PIP. DOS always signals its readiness to accept commands by typing the character $ on the system teletype; PIP requests a command using the > character.

Typing the character <*cntrl C*> will always return control to DOS. It is important to begin any session by turning OFF (unlit) all of the pushbuttons below the display and typing <*cntrl C* > and LOGOUT on the system teletype.

The underlined commands below will transfer the PATH binary files from DECtape 155 to the scratch disc area. To ready the tape, mount it on drive unit *u* and set the WRITE switch to LOCK. When typing, command lines are terminated by <*cr*>. Comments in the example are preceded by the character / and should not be typed. DOS messages will appear on the display if VT is ON.

```
$ <cntrl C >          / GET DOS monitor
$ LOGIN SCR           / LOG IN to scratch disc area.
                      / SCR could be replaced by operator's UIC
$ KEEP OFF            /.Clear I/O assignments
$ PIP                 / Get Peripheral Interchange Program

  DOSPIP V6A
> T DK ← DTu PATH XCT (B)   / transfer XCT file
> T DK ← DTu PATH XCU (B)   / transfer XCU file
> T DK ← DTu PATH LØ1 (B)   / transfer link files
> T DK ← DTu PATH LØ2 (B)
      .          .
      .          .
      .          .
> T DK ← DTu PATH LØ5 (B)
> T DK ← DTu PATH MNB (B)   / Transfer MENDEL binary
> <cntrl C>
$ DOS-15 V2A
```

If in doubt at any point, ask an operator at the 1906A console for help. Please note that GAPs *cannot* be run directly from DECtape. Once the files are on the disc the DECtape is no longer needed and may be switched to LOCAL, rewound, and returned to its rack.

Next, the operator should consult the documentation for any particular I/O requirements the GAP might have. These could include mounting a particular magnetic tape or DECtape; switching on the BSI, VCS3 synthesizer, or DMAC pen follower. If at all in doubt, let a 1906A operator set up the peripherals.

It will often be necessary to make certain device allocations to the DOS monitor's DEVICE ASSIGNMENT TABLE (DAT) before loading the GAP. PRTE uses only default assignments, but the application may have special requirements. Including the PRTE device assignments for information only, the proper DAT slot assignments for PATH are:

```
$ A   TTA 4       / PRTE - KEYBOARD INPUT
$ A   DK  6       / PRTE - DISK
$ A   VTA 1Ø      / PRTE - DISPLAY
$ A   VWA 11      / PRTE - SPARKPEN
$ A   NON 16      / PATH - DMAC(not used)
```

DAT slot -3 is used by PRTE for error message output but cannot be ASSIGNed by the operator.

Finally, the operator must turn off the VTØ4 display by typing:

```
     $  VT OFF
```

and ensure that the VW01 sparkpen is ON and the LK35 keyboard is OFF.

The system overlay program, EXECUTE, is used to load the GAP into core and start it running. In DOS the operator should type the characters E *<sp>* followed by the GAP name. At this point the teletype will output two linefeeds and execution will begin. To load PATH for example, type:

```
     $  E PATH <cr>
```

If at this point DOS types out the message:

```
     IOPS 4
```

it means that some device is not on-line or is not switched ON. Ready the peripheral and type *<cntrl R>* on the teletype. Other IOPS errors may occur because of mistyping the DAT slot assignments or forgetting to transfer one of the files from DECtape to disc. Try the entire sequence once more, if in doubt, and then summon a 1906A operator for assistance. Typing *<cntrl C>* will bring back DOS with the default DAT slot assignments.

When the GAP has been loaded, the message:

```
     >PIGS Vn
```

will be output. At this point the name of the MNB file should be input. For PATH, simply type:

```
     >PATH <cr>
```

When the PIGS display appears on the VTØ4, PRTE is ready to accept commands.

1.2 Screen Layout

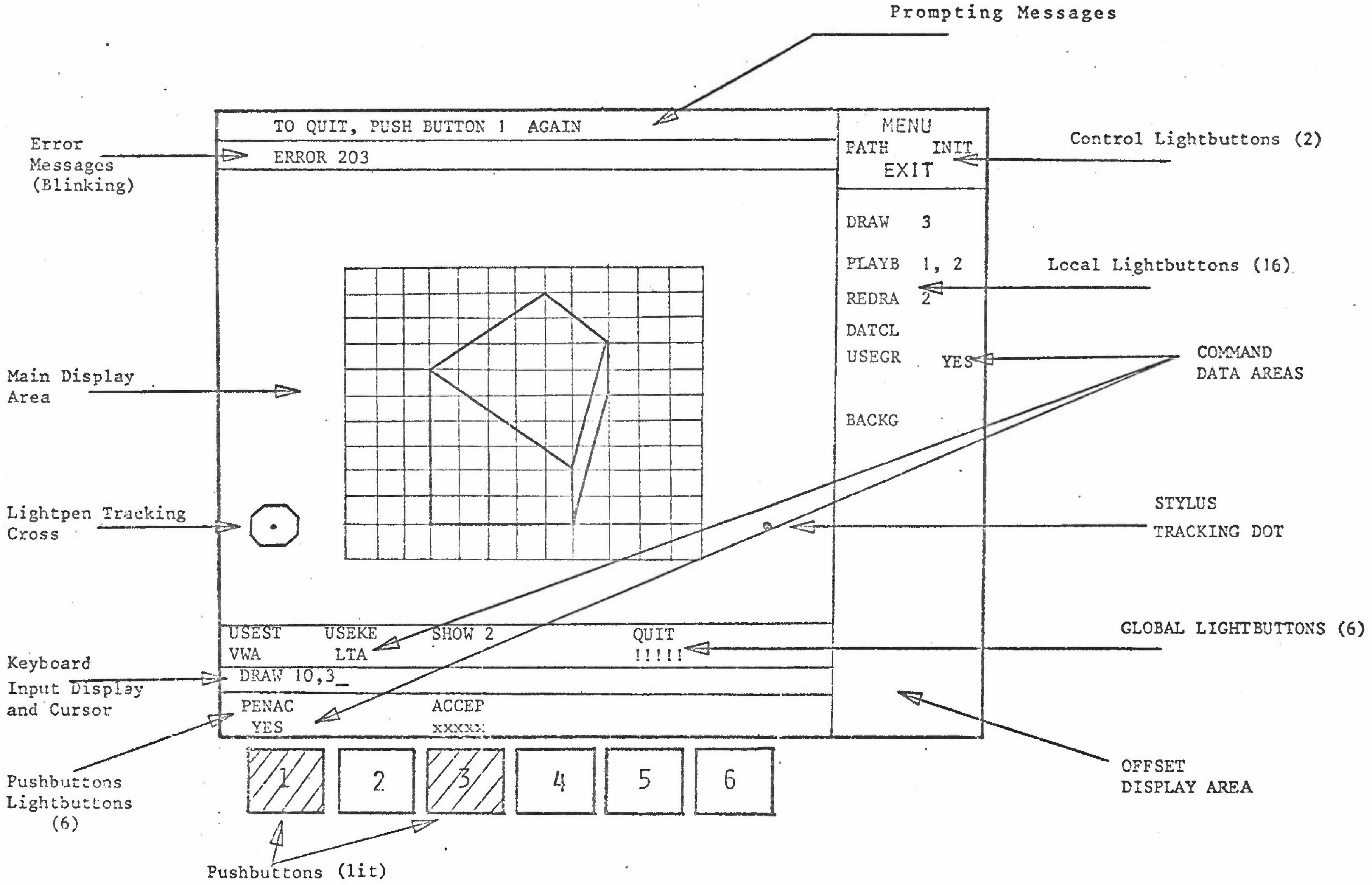
The layout of any display using PRTE has a standard form with only a few variations. Figure 1.1 shows the initial display put up by PATH.

The standard display used by the PIGS run time environment consists of a large enclosed central workspace bordered on the top, bottom, and right sides by message and command areas. The displayed text may be composed of large or small characters of the designer's choice. The large characters are easier to read and photograph but use up more of the workspace and allow fewer characters per item.

Each of the areas bordering the central workspace has a specific function. Beginning at the top of the screen and working clockwise, the topmost area is for operator *prompting*. Messages containing possible courses of operator action may be placed there by the GAP. These will change during the course of interaction and are of particular aid in learning to use a new package.

Just below the prompting space is the *error message* area. When an error occurs or is about to occur in a procedure, the teletype bell will ring and a flashing message will appear in this space. The prompting area may suggest a recovery procedure at this time. The text will disappear when the condition is corrected or after the next operator command.

FIGURE 1-1 - PIGS DISPLAY



To the right of the error and prompting messages is the menu *control* area. Two text strings appear permanently in this space. The name of the current command set will appear below the characters MENU. The name of an alternative menu will appear below EXIT. These commands, unlike messages, are sensitive to selection with the active stylus. Note that a small *command data* area appears next to each name. Such areas may contain information about the adjacent command and often change during the course of interaction.

The long vertical region below the menu control space is the *local command* area. Commands appearing in this region change from menu to menu. As many as sixteen names and data areas may appear in the local command area. Like menu names, local command names may be selected using the stylus.

At the very bottom of the screen is the *pushbutton command* area. One command name may appear over each button with a data space *below* it. Pushbutton names are stylus-sensitive.

Monitoring of typed commands occurs character by character in the *keyboard input* area just above the pushbutton names. The last typed text remains displayed until the first keystroke of a new command. Like the error and prompting messages, this text is not stylus sensitive.

The last PRTE display region has the same appearance as the pushbutton control area but is located slightly above the keyboard input text. The *global command* area consists of six command names which remain constant regardless of the menu name. Each command name is stylus-sensitive although the matching data space below it is not.

Usually, a graphics application confines its display to the central enclosed workspace. It is possible, however, for a GAP to turn off the standard PRTE display and use the entire screen as a work area.

1.3 Commands

In the course of interaction with an operator, the PIGS run-time environment may receive requests from a variety of peripherals. These orders are called *commands*. Their text names form a language for communication between man and machine. When displayed, these names are called *lightbuttons*.

In constructing a GAP, the designer writes a subroutine for each function in the package and assigns a command name to it. When PRTE receives a command it causes execution of the matched subroutine in core and seeks the next command.

1.4 Command Sources

PRTE spends most of its time looking for orders from its command sources. This going from door-to-door is aptly referred to as *polling*. The possible sources of commands are:

(1) Lightbuttons
(2) Pushbuttons
(3) Real Time Clock
(4) Keyboard

Quite often only one or two of these devices is appropriate for the type of interaction desired. The designer decides which sources will be polled when a particular set of commands is displayed. These sources are referred to as *active*. Attempted input from an inactive source is ignored.

1.5 Command Selection

Corresponding to each possible PRTE command source are one or more *source devices*. In most cases only one of these hardware input devices may be active for each of the four command sources.

(1) Lightbuttons

Lightbuttons, including the displayed pushbutton names, may be selected using either the lightpen or the VW01 sparkpen (abbreviated LPN and VWA). PRTE signals the operator that a lightbutton has been touched, or *hit*, by blinking the button on and off for about ½ second. This allows the operator to move the stylus away from the button and thus avoid executing the command more than once. Occasionally, however, it is desirable to repeatedly execute a command for as long as its corresponding lightbutton is being hit. An example might be a command to rotate a displayed object one or two degrees at a time. Repeated hits on such a button would produce the effect of continuous rotation of the object.

It is up to the GAP designer to decide whether the lightpen or sparkpen is the active source device. He may provide a command to switch between the stylii or he may not. If there is such a command, the sparkpen must be turned ON when changing source devices, otherwise an IOPS 4 error will occur. If such an error does occur, simply switch the pen ON and type <*cntrl R*> on the system teletype to continue; subsequently the sparkpen may be turned OFF whenever desired.

If there is a command to change stylii, the active source device should be obvious from the associated command data area. Assuming the sparkpen is active, it will emit a harmless, continuous spark when the switch on the tablet is in the ON position. Like spark plugs in a car, the electrodes of these pens corrode and wear out. Treat the pen gently and turn it off when not in constant use.

The VW01 tablet senses the position of the sparkpen using foil microphones positioned along two edges of the flat surface. These microphones are used to time each spark and should not be obstructed by any object. The tablet is normally set up for right-handed people with the microphones to the top and left sides. Left-handed users should ask a 1906A operator to set the left-hand switch on the sparkpen logic and re-orient the tablet. Paper may be placed on the surface of the tablet, but this is not necessary.

The stylus itself should be grasped with the white line near the tip facing upwards. The proximity of the pen to the tablet surface is detected as three states: *far*, *near* and *touch*. Slight downward pressure on the stylus against the tablet will cause the biro to retract and the pen to enter touch mode. Near mode is entered when the stylus is lifted slightly; 3 or 4 inches above the surface of the tablet the sparkpen is in far mode and its position cannot be accurately sensed.

When the stylus is in near or touch mode, PRTE maps its location on the tablet surface into a relative position on the CRT and displays a bright dot called the *tracking dot* at this point. The operator will quickly find it natural to watch the display and not the stylus while drawing or pointing.

To select a particular lightbutton, first imagine that an invisible rectangle surrounds the text name. Move the stylus in near mode until the cursor lies within this rectangle; pressing the pen down at this point will cause selection. When the lightbutton begins to blink, lift the pen up into near mode to avoid multiple lightbutton hits.

If the lightpen is the active stylus, a slightly different selection procedure is necessary. The lightpen detects light from the CRT directly using a photocell. The sensitivity of this cell is regulated by a small knob just below the screen on the left. This knob should be turned all the way clockwise for maximum sensitivity. To select a particular lightbutton, point the pen at the text name and press the button on the top of the stylus all the way down. When the text begins blinking, release the button to avoid multiple hits.

(2) Pushbuttons

As a command source the pushbuttons are unique in that they may be selected by either the active stylus or by one of the six blue contact buttons beneath the screen. Each button may light up when pressed, or it may not, depending on the application program.

Pushbuttons are also unique in that they are commonly used to control a GAP subroutine within the execution of a command. In this case, the pushbuttons will probably not be sensitive to selection at polling time. In PATH, for instance, pushbuttons 1, 2, and 3 (left to right) are used to control the active stylus within the commands DRAW and DRAWB but are not sensitive at other times.

(3) Real Time Clock

The 50 Hz clock may be used by GAP subroutines to schedule the repeated execution of commands at some time interval; a good example would be a command to dump an application database regularly. A GAP may signal to the operator that a scheduled command is being executed by placing a message in the prompting area.

(4) Keyboard

Like the lightbuttons, there are several alternative devices which may be used as the keyboard command source: the LK35 keyboard (LTA) on the display console or the system teletype (TTA). A command for switching between devices may or may not be available, depending on the particular GAP.

Characters typed on the active device are always displayed in the keyboard input area of the CRT. Optionally, LK35 input may be echoed on the system teletype as well; a command is not selected until carriage return or altmode is typed. At this time if there is a lightbutton displayed for the command, it will blink. Characters may be deleted from the input string by typing <*rubout*>; the entire string is cleared by typing <*cntrl U*>.

After typing the first character of the new command, and before terminating it, no other command source is active. Clearing the input string with <*cntrl U*> causes polling to resume, however, with no command having been executed. Also, typing <*cntrl P*> on the system teletype may cause return to the PRTE polling loop - *but only try it in an emergency*.

There are several peculiarities of the keyboards which have not yet been cured and may cause some annoyance.

The most important of these is interference between the VW01 sparkpen and the LK35 keyboard. When both are running, electronic noise from the spark is occasionally picked up by the keyboard circuitry and interpreted as character input When this happens all other command sources become inactive and, most noticeably, the cursor on the display no longer reflects sparkpen movement. To clear this condition turn the tablet OFF and type <*cntrl U*> on the LK35. Then turn the keyboard OFF using the small toggle switch on the right-hand vertical side of the VT04 console table; turn the tablet back ON to select commands using the sparkpen. Work is in progress to cure this hardware fault.

The second peculiarity also concerns the LK35 but is only mildly irritating. The keyboard has a shift *lock* key which may be inadvertently hit. There is no way of knowing if the device is locked into upper case except by typing. To clear the condition, hit the shift key and type <*cntrl U*>.

The last keyboard problem occasionally occurs on the system teletype when one has switched from TTA to LTA and then attempts to return to TTA. At this point all polling will cease, as in the sparkpen interference situation, until any key is struck on the system teletype. This clears a handler conflict; normal polling will resume with TTA active and the keyboard input string empty. There is no way to cure this software fault without altering the DOS operating system.

1.6 Command Syntax

When an operator selects commands by typing on the active keyboard source, certain punctuation rules, like terminating the line with *<cr>* or *<altmode>*, must be followed. These rules - or *syntax* - are specified in a formal manner in the Appendix; here they are illustrated by example.

Sometimes, in designing a command, it is convenient to closely associate with it a set of parameters called *arguments*. Typed after the text command name, these values are made available to the GAP subroutines while the command is being executed. In PATH, for example, typing the string:

```
DRAW, 1 <cr>
```

causes PRTE to pass along the value to the application subroutine; cel 1 is opened for sketching.

The most general form of a typed command is a command name followed by one or more arguments and terminated by *<cr>* or *<altmode>*. As in the simple example above, the command name must be separated from its arguments by a comma. Similarly, each argument is separated from its successor by a comma, as in:

```
PLAYBACK, 1, 2, 50 <cr>
```

Note that there is no punctuation between the last argument and the terminator *<cr>*. A command may have no more than 14 arguments; blanks are ignored, except within text strings.

The arguments to a command need not be explicitly specified, however. An argument may be omitted by typing only the trailing comma in its place. The second argument is omitted in the example below:

```
PLAYBACK, 1,, 50 <altmode>
```

Arguments which have been omitted are assigned standard, or *default*, values by GAP subroutines. If an argument which *must* be specified is omitted an error message will appear on the display and the command will be ignored.

Arguments may also be omitted by truncating the argument string with a *<cr>* or *<altmode>*. All three arguments in the previous example assume default values if:

```
PLAYBACK <cr>
```

is typed. Exactly the same thing would happen if the command PLAYBACK were selected using the lightbutton source.

A final consideration about the syntax of commands is the number of characters which must be typed to completely specify a command name.

With the small text display, PRTE matches a 5-character name if it has been placed in *abbreviate* mode by the GAP. Otherwise it looks for a 9-character name. PRTE is always in abbreviate mode if the large text display is being used; more than 5 characters may be typed but they will be ignored from the sixth onwards. It is never necessary to pad a genuinely short command name in order to make it 9 characters long.

1.7 Argument Types

As detailed above, a valid command consists of a command name followed by a string of arguments separated by commas. By the time these arguments reach a GAP subroutine, the only distinction made in argument types is between text strings and numbers. To allow greater precision all numbers are stored in double precision floating point format. The total length of all string arguments to a command must be less than 70 characters.

String arguments may be entered in one of two basic formats: unquoted or quoted. Unquoted strings must begin with an alphabetic character and are terminated by blank, comma, <*cr*>,or <*altmode*>. Quoted strings are begun and terminated by a pair of single or double quotes. Blanks, commas and any other character except <*altmode*> or <*cr*> may be embedded in quoted strings. Note that command names, which are simply strings, may be unquoted or quoted. Therefore, if a command is displayed with embedded blanks or a non-alphabetic initial character, its name must be quoted when selecting it via a keyboard device. Below are examples of valid and invalid string arguments.

Valid strings:

```
    DRAW75
    'SELECTΔ3'
    "FINDΔ4"
    '6"ΔHOLE'
    ΔΔBAR61Δ
    '...PEG'
    "1'ΔHOLE"
    '6"ΔDIAMETER'
```

Invalid strings:

```
    .DRAWB
    PLACEΔBELOW
    "1"ΔHOLE"
    '6"ΔDIAMETER"
```

Number arguments have a very flexible format consisting of three parts: a radix indicator, a signed mantissa, and a signed exponent. If a number is preceded by the character #, octal radix is assumed. After the radix indicator a sign may appear followed by the mantissa. The latter may have only an integer part, only a fractional part, or both separated by a full stop. Exponentiation may be specified after any form of the mantissa by typing the character ↑ followed by a signed integer; if octal radix has been indicated the base of the exponent is 8. As they are stored in double precision format, the value of number arguments must be less than 10^{75} and greater than 10^{-75}; the accuracy is 33 bits (9 digits).

The complete format for a number argument, although powerful, is lengthy. In practice only some form of the mantissa need be typed. If the character # does not precede the number, decimal radix is assumed; the default for omitted signs in mantissa or exponent is +. Below are some examples of valid and invalid number arguments.

Valid numbers:

```
-123456789
 12345.6789
 13↑-40
 40.↑13
 +ΔΔ3.14159↑2
 #-77.1↑1
```

Invalid numbers:

```
1234567890D1
100↑74
1.5↑36.4
#999
```

In summary, there are four major rules for composing syntactically well-formed commands:

(1) The body of a command consists of a command name followed by arguments separated by commas.
(2) The command name is simply a text string and must be quoted if it contains embedded blanks, commas, or begins with a non-alphabetic character.
(3) Arguments may be numbers or strings, or they may be omitted.
(4) A command is terminated by *<cr>* or *<altmode>*.

Below are included some examples of valid and invalid commands:

Valid commands:

```
DRAWB,3 <cr>
DATACL  <altmode>
DRAW75, '6"ΔHOLE' , "AT",360, 1,023↑+3 <cr>
DRAW75,,,#1000 ,#10↑2 <cr>
"1'ΔHOLE",DRILL8,'FORΔOIL' <altmode>
```

Invalid commands:

```
NOCOMMA 3 <cr>
MUST,QUOTE, 6"ΔHOLE  <altmode>
TOOMANY,1,2,3,4,5,6,7,8,9,10,11,12,13,14,OOPS! <cr>
BADARGS, 'GOBBLE, #94 <cr>
```

1.8 Menus

Once source devices and selection procedures are understood, it is still necessary to know at what times a command may be typed. In general, if the name of the command in question is displayed on the screen it may be selected with any active device. However, because of space limitations on the display and in core, it is often not possible for all the commands of an application program to be available for selection at once.

For this reason related commands are grouped together into sets called *menus*. When a particular menu is *active* (or *current*) its name will appear in the control area of the screen beneath the fixed characters MENU. A set of local commands and pushbutton names peculiar to the current menu will be displayed in their respective areas of the screen. Any of these commands are available for operator selection. Normally the commands contained in a given menu are sufficient to complete some small portion of the application task.

Usually, the menus comprising a GAP are hierarchically organised: a menu may contain commands which cause new menus to become active. For example, the first menu which appears on the screen when PATH is loaded contains the name of another menu in the GAP: BACKG. Selecting this command will cause the appropriate menu to appear on the display along with its associated local and pushbutton names. Simultaneously, the name of the top level menu, PATH, will appear in the control area below the fixed string, EXIT. Selecting PATH will cause a return to the top level menu.

In summary, the control area of the display has two principal functions: naming the currently active menu and allowing return to a higher level menu. As in the menu PATH, however, *any* command may cause a new menu to become active. Usually the name of such a command is identical to the name of the menu it makes current. There is no question of missing a menu change,because on activating a new menu the old command names will disappear from the display for a short time, making the switch unmistakeable.

1.9 Global Commands

Local and pushbutton commands may only be selected when they appear on the display as part of the current menu. Global commands, by contrast, are always available regardless of the active menu and need not all be displayed.

As many as six of the set of global commands for a particular GAP may appear on the screen just above the keyboard display. These lightbuttons function normally, but are not affected by menu changes. Like other commands they may be selected using any active device.

In addition to these six, there may be any number of non-displayed global commands which cannot be selected using lightbuttons, but may be chosen using any other source device. The operator should refer to the particular GAP documentation for the names of non-displayed globals. There are several such commands in PATH: SKEDL, for instance, allows scheduling of commands for selection by the real time clock.

1.10 Error Messages

A complete list of error messages is given in Appendix K of this manual. There are, sadly, five separate categories of errors which may occur during the execution of a GAP:

(1) GAP errors
(2) PRTE errors
(3) FOG errors
(4) OTS errors
(5) IOPS errors

GAP, PRTE, and FOG errors all appear on the display in the error message area and cause the teletype bell to ring. None of these errors should cause the application to terminate. In the case of PRTE and FOG errors, control always returns to the command polling cycle. Program control after a GAP error may return to the polling sequence or to the application subroutine. In the latter case the prompting area may contain suggestions about correcting the condition. Error messages disappear upon selection of the next command.

GAP documentation should contain information about the errors its programs issue. Error numbers less than 100 are generated by PRTE, in the range 100-199 by FOG, 200 upwards by the GAP itself.

OTS and IOPS errors normally cause the application program to terminate and control to return to the DOS operating system. These errors will appear on the system teletype and, except for IOPS 4 (device not ready), should be noted in the system log book. Such errors ought not to occur in a GAP which has been debugged.

1.11 Quitting

Having finished a task using an applications program, there are two ways to return control to the operating system: by package command, or by typing *<cntrl C>* on the system teletype. If there is a session terminating command in the package it is best to use it as there may be open files which need to be closed.

2. WRITING A MENDEL PROGRAM

2.1 Introduction

The graphics application designer uses the MENU DEFINITION LANGUAGE, MENDEL, to describe the organisation of commands and menus to the PIGS run time environment. It may also be used by an operator to edit the command structure of an application to facilitate his style of interaction or to meet particular problem demands. As in Chapter 1, most examples in this chapter will be drawn from the graphics application, PATH. The MENDEL description of PATH appears in Appendix A.

2.2 MENDEL Commands

MENDEL commands are compiled by the editor-assembler to create or modify a binary file on disc or DECtape. This file, with extension MNB, is accessed by the run time environment during initialisation and when changing the active menu. The mechanics of loading and running the MENDEL editor-assembler are described in Chapter 3. Suffice it to say here that commands may be typed to the editor-assembler one at a time, or read from an ASCII text file on some storage device.

Regardless of whether commands are typed directly or read from a file, each has the same form; MENDEL commands use the same syntax as PRTE commands, described in Section 1.6. A single difference is that comments may appear in a MENDEL program by preceding the text with the character /. The following sample statements are accepted by the editor-assembler:

```
/THIS IS A COMMENT. <cr>
COM,MNDEC,-1,DO,YYMND /DEFINE COMMAND MNDEC <cr>
MENU,MDL <altmode>
```

Note that a comment by itself is a well-formed statement ignored by the editor-assembler.

2.3 Command Modes

MENDEL commands may be issued with one of two purposes: to create a new binary MNB file or to edit an existing file. Consequently, the first command to the editor-assembler sets the *mode* in which subsequent statements will be obeyed. Most MENDEL statements are valid in either create mode or edit mode; the few which are not are marked in the list of commands in Appendix I.

The principal difference between the two modes is not in the function of commands, but in the order in which they may be executed. Edit mode commands may be issued in any order, while create mode commands must, for efficiency reasons, follow a rigid sequence. A second difference is that MNB file size, which determines the number of commands and menus which may be defined in an application, may be altered only in create mode. It is possible that a later version of PRTE will itself contain the MENDEL editor, thus allowing run time changes in the active MNB file.

NGLOBLS - Maximum number of global commands contained in the application. Again, add a few spares.

(2) BIGBT

The BIGBT command is valid only in create mode as it alters the structure of every command in the MNB file. If this statement is encountered by the assembler, the large lightbutton characters will be displayed and only 5 characters of application command names matched. If the BIGBT statement is not encountered,the small display size will be used.

(3) ABREV,ALOGIC

This command sets the initial command name matching size of PRTE to 5 characters instead of 9 characters. Only 5-character command names are matched with the large display.

ALOGIC - Optional logical argument. Must be the text string TRUE or FALSE.
If omitted, TRUE is assumed.
TRUE - Only the first 5 characters of command names need be specified.
FALSE - Entire command name must be specified (max 9 chars).

(4) KEYB,AKEYBD,AECHO

KEYB selects the initially active keyboard source device and whether or not typed characters are echoed on the system teletype. If KEYB is not encountered, the LK35 will be the active keyboard with no echo.

AKEYBD - String argument specifying the active keyboard.
LTA - LK35 keyboard
TTA - System teletype
If omitted, LTA is assumed.
AECHO - String argument specifying whether or not LTA input is echoed on TTA.
ONLY - Do not echo
ECHO - Echo
If omitted, ONLY is assumed.

(5) STYLS,ASTYLS

Defines the initially active stylus source device to PRTE. If the command is omitted, the sparkpen will be used.

ASYTLS --String argument specifying the active device.
VWA -Sparkpen
LPN -Lightpen
If omitted, VWA is assumed.

(6) DELAY,NMILSEC

Sets delay time after command selection and before command execution. During this time any associated lightbutton will wink. This delay does not apply to commands marked for immediate execution. (See ADOCODE argument of COM command). A delay of 25∅ milliseconds is assumed if the DELAY command is not encountered.

NMILSEC - Number of milliseconds to delay before execution. If omitted, 25∅ is the default value.

(7) SAVE,ALOGIC

Each menu described using MENDEL contains information, such as command data area text, which may alter at run time. When the active menu is changed new data is lost if the copy of the menu in core is not written out over its corresponding MNB block. If the editor-assembler encounters the SAVE command the core image of any menu flagged using the SVMNU statement will be written over its corresponding MNB file definition before the menu becomes inactive. (The SVMNU command is described in context 6, below). Time and effort in maintaining current display data is saved, but the disadvantage of permanently altering the MNB file is incurred until the MENDEL source is reassembled. If the SAVE command is omitted, SVMNU requests are ignored and no menus are saved.

ALOGIC

TRUE - Enable saving of flagged menus.
FALSE- Disable saving of flagged menus.
If omitted, TRUE is assumed.

2.6 Menu Declaration: Context 2

Each menu defined using MENDEL is allocated a single data block in the MNB file and given a unique 9-character name. The names and block numbers are kept in a permanent *m*-block MENU ADDRESS TABLE (MAT) beginning at block 2 of the MNB file. Each block of the table may contain as many as 41 entries. The number of blocks actually allocated for the MAT is determined from the NMENUS argument of the CREAT command.

The purpose of context 2 is to assign block addresses to menu names in the MAT before any application command definitions actually occur. Because a menu change may be specified in the definition of a command, it is helpful to know the block numbers of all menus. Declaring the menu names in context 2 lets the assembler completely process each command in a single pass.

Menu names are stored and matched as 9-character strings regardless of the command matching mode set by ABREV or BIGBT. In subsequent contexts referencing a menu name which has not been declared will cause an assembly error. Menu names which are declared but never defined will not be flagged as errors.

(8) MNDEC,AMENU,AMENU,...

There is only one valid command in context 2: MNDEC ends context 1 and enters a list of menu names in the Menu Address Table. The command must have at least one argument and may have as many as will fit in a 70-character line; the MNDEC command may be repeated as many times as desired within context 2.

AMENU - Menu name (maximum 9 characters). Quoted strings are allowed. These names do not appear on the run-time display.

2.7 Subroutine Declaration: Context 3

Menu definition using MENDEL requires matching an application subroutine name with each command name so that the run-time environment can cause execution of the appropriate procedure in core. These subroutine names are kept in an *s* block Subroutine Name Table (SNT) immediately after the MAT in the MNB file. After completing the processing of a create mode program, the assembler uses the SNT to construct the Jump Table, a relocateable binary file which, when loaded, contains the entry point address of each application procedure.

As the ordering of the names in the SNT corresponds to the ordering of entry point addresses in the Jump Table, only the index in the SNT of the procedure to be executed need be stored with each command. When a particular command is selected PRTE picks up its associated SNT index and jumps to the procedure indirectly via the entry point address in the Jump Table.

The purpose of context 3 of a create mode MENDEL program is to declare all subroutine names to be included in the SNT. The 1 to 6-character names are stored 62 entries per block, the number of blocks being determined from the ASUBBR argument of the CREAT command. The order of specification of the procedure names is unimportant, but references in later contexts to undeclared subroutine names will cause an error message.

(9) SBDEC,ASUBBR,ASUBBR...

There is only one valid command in context 3. The SBDEC command terminates context 2 and causes the MAT to be written out to the MNB file. At least one argument to SBDEC must be given, but as many as desired may be specified. The command may be repeated within context 3.

ASUBBR - Procedure name (maximum 6 characters).

2.8 Global Command Definition: Context 4

Contexts 4, 5, and 6 of a MENDEL program are concerned with command definition. Each displayed or non-displayed application command is represented by a 10 word *node* whose structure is given in Appendix E. Command nodes are grouped 24 per block with a 10-word header node to form *menu blocks*.

The global command nodes belonging to a particular GAP reside in *g* contiguous menu blocks following the SNT. The number of blocks allocated is determined by the NGLOBLS argument to the CREAT command. The purpose of context 4 of a MENDEL program is to enter nodes in those blocks representing the global commands of the application.

Command nodes are entered in a menu block at the current *command cursor* position. In a create mode program, entry into context 4, 5, or 6 causes the cursor to point to the first command node of the first menu block allocated. There are MENDEL statements in both create and edit modes which re-position the cursor. Since this allows command nodes to be skipped, unaccessed nodes are always preset to the null command.

The first six nodes of the first global menu block define commands to be displayed in the global command area. The remainder of the nodes describe non-displayed global commands. Unused displayed commands may be skipped by inserting COM statements with no arguments.

There are two valid MENDEL statements in the global command definition program unit:

(10) GLOBL

With no arguments, this command ends context 3 and causes the SNT to be written out to the MNB file.

(11) COM,ANAME,NARGS,ADOCODE,ASUBBR,AMNUCODE,AMENU,ADATA

The COM statement describes an application command to the PIGS run-time environment. Information about the GAP command is entered in the current command node and the cursor is incremented. COM has 7 arguments, any of which may be omitted. If all are omitted, a null command is entered in the current node. In context 4, if the current menu block is filled, the cursor is automatically positioned to the first command node of the next menu block allocated. An error results if all allocated blocks are already full. The arguments to COM are:

ANAME - 5 or 9-character command name. Blank if omitted.

NARGS - Maximum number of arguments associated with the command; a value of -1 means an indefinite number may be specified.
If omitted, Ø is assumed.

ADOCODE- One of 4 strings specifying how the associated application procedure is executed:

- DO - Call procedure after interval set by the DELAY command.
- DONOW- Call procedure without any delay.
- DONT - Command inactive until activated by the GAP. When activated, operates like DO. Inactive commands are not displayed until activated.
- DTNOW- Command inactive. Operates like DONOW when activated by the GAP.

If omitted, DO is assumed.

ASUBBR - Name of associated application procedure. The name must have been previously declared using the SBDEC command. If omitted, no procedure is executed when the command is selected.

AMNUCODE- One of 4 strings describing protocol if a menu change is to occur. Execution of the exit procedure of the current menu or the entry procedure of the new menu may be specified.

Code	Menu protocol
MENU	Exit, new menu, entry
ENTER	New menu, entry
EXIT	Exit, new menu
GO	New menu

If omitted, MENU is assumed.

AMENU - Name of new menu to be activated. Must have been previously declared using the MNDEC command. If omitted, no menu change occurs.

ADATA - 5 or 9-character string to be initially displayed in the command data area. Blank if omitted.

2.9 Argument Getting: Context 5

The function of this context is unimplemented in PIGS V2. However, the single command must be present to close context 4:

(12) ARGET

Closes context 4 and causes the current global menu block to be written to the MNB file.

2.10 Menu Definition: Context 6

The bulk of a create mode MENDEL program is concerned with describing the control, local, and pushbutton commands composing the various menus. Context 6 basically consists of a group of COM statements for each menu named in the MAT. Each *menu definition* begins with a MENU statement and ends with a subsequent MENU statement or the termination of context 6. The order of occurrence of the definitions is irrelevant.

A menu definition may contain three types of commands: header description statements, cursor movement statements, and command definition statements. The effect of the statements is to generate header entries or command nodes in the single menu block assigned by the MNDEC declaration. Any of the three types of statements may be omitted except the MENU command. If all other statements are omitted, a null menu is created. For obvious reasons it is best to include at least a menu exit command.

The ordering of both header description and cursor movement statements within a menu definition is not crucial. The order of the command nodes generated in the menu block is crucial. The programmer is well advised to follow the sequence illustrated by PATH MDL in Appendix A.

As shown in Appendix E, the first two nodes of the associated MNB menu block describe the entry and exit commands displayed in the control area of the screen below MENU and EXIT. The application procedures associated with these commands double as the entry and exit procedures for the menu, respectively. These subroutines may, of course, be omitted. If a hierarchic structuring for the menus is desired, the exit command should cause activation of the next higher, or *father*, menu. Descent to lower *brother* menus is usually implemented using local commands.

Some care should be taken to ensure that the menu change protocol caused by the exit command does not cause an untimely initialisation of an application data base. Remember that the MENU argument to the COM statement causes execution of both the exit procedure of the old menu and the entry procedure of the new menu. Since the entry procedure to a father menu will occasionally initialise a GAP data base, and because selecting an exit command will cause the associated exit procedure to be executed anyhow, the GO argument to the COM statement is ordinarily used to specify the menu change for the exit command.

The next 6 nodes in the MNB menu block define the displayed pushbutton commands, 1-6, left to right. It is often convenient to use the pushbuttons for simple interaction within the execution of some GAP command. If this is the case, the pushbuttons may be labelled by omitting the ASUBBR and AMENU arguments in COM commands for the appropriate nodes.

The remaining 16 nodes in the MNB menu block define the local commands, displayed at the right-hand screen edge. All 16 nodes need not be used; it is useful to separate the commands into subgroups using null commands, and no loss of polling efficiency is incurred. In writing a menu definition it is obviously important to know the current cursor position. When a MENU statementis encountered in a create mode program, the cursor is initialised to point to node 1, the entry command. Several cursor positioning commands are included in context 6 for clarity and convenience. It is always best to use the ENTER,EXIT,PUSHB, and LOCAL statements as illustrated in Appendix A.

All context 6 commands are described below. The three header description commands are:

(13) MENU,ANAME

The MENU command begins a menu definition in context 6. Any previously defined menu block is written to the MNB file. The command cursor is initialised to point to the first node of a menu block filled with null commands. At least one menu definition must be present in a create mode program.

ANAME - 9-character (maximum) menu name previously declared using the MNDEC statement. May not be omitted.

(14) DSABL,ADEV,ADEV...

The DSABL command specifies which command sources will *not* be polled when the menu being defined is activated by PRTE. The DSABL command need not appear; all unspecified sources will be polled. DSABL may have an indefinite number of arguments and ordering is irrelevant.

ADEV - One of four strings specifying which of the command sources is to be disabled.

KEYB	-	Keyboard source
LTBUT	-	Lightbuttons
PUSHB	-	Pushbuttons
CLOCK	-	Real time clock

(15) SVMNU,ALOGIC

Controls the saving of the current menu by PRTE. The core image of the menu will be written to the MNB file on activation of a new menu if the SAVE statement was encountered in context 1.

ALOGIC -Optional logical argument
Must be the text string
TRUE or FALSE.
If omitted, TRUE is assumed.
TRUE - save the state of the menu currently being defined
FALSE - Do not save
If omitted, TRUE is assumed.

The four cursor movement statements are, in recommended use order:

(16) ENTER

Positions the command cursor at command node 1, the entry command.

(17) EXIT

Positions the cursor at node 2, the exit command.

(18) PUSHB,NBUTTON

Positions the cursor at the node corresponding to pushbutton NBUTTON (nodes 3-8).

NBUTTON Number of pushbutton, 1-6, left to right.
If omitted, 1 is assumed.

(19) LOCAL

Positions the cursor at the first local command node, 9.

The only other legal command in context 6 is the COM command. Its arguments and function are as described for context 4.

2.11 Program Termination: Context Ø

The final unit of a create mode MENDEL program has the function of terminating assembly and naming a starting menu for PRTE. The END command must be present in a create mode program.

(20) END,AMENU

Terminates context 6 and writes out the last menu block defined. After the MNB header block is recorded the file is closed.

AMENU - Name of starting menu to be ENTER'ed by PRTE at run time. If omitted, the first menu declared by an MNDEC statement is assumed.

2.12 Edit Mode

If an MNB file already exists, having been constructed using a create mode program, MENDEL may be used to edit initialisation parameters, application commands, and menus. If extra space in the MNB file is available, it is possible to add new commands and menus. All create mode statements except CREAT and BIGBT are valid in edit mode as well.

On beginning an edit, MENDEL reads into core the old MNB header block, MAT, and SNT. Subsequent GLOBL and MENU edit commands cause the associated MNB menu blocks to be retrieved. On completion of *each* edit command, the original data blocks are overwritten with the altered data blocks in core. Thus in edit mode only, contexts may be entered in any order desired. In edit mode the GLOBL and MENU commands position the cursor at the last defined command node +1.

The following two commands begin and end a MENDEL edit. They may not appear in a create mode program.

(21) EDIT,ANAME

Begins a MENDEL edit. The original MNB header, MAT, and SNT are retrieved.

ANAME - Name and extension of MNB file to be opened for editing. If the extension is omitted, MNB is assumed. The argument may not be completely omitted.

(22) FIN

Terminates an edit by closing the MNB file.

The following commands are valid only within edit mode contexts 4, 5, or 6:

(23) POS,NRELATIVE

or

(24) POS,ACOMNAME

This command repositions the command cursor in the current menu block or group of menu blocks. In global context all allocated menu blocks are available - block boundaries are ignored. In context 6 only the single menu block allocated to the current open menu is available.

NRELATIVE - Number of commands to move the cursor forwards or backwards.
NRELATIVE may be negative. If omitted, ∅ is assumed.

ACOMNAME - 5 or 9-character (maximum) application command name to be searched for by the editor-assembler. If found, the cursor is left pointing at the node defining the named command. If not found, the cursor position remains unchanged.

(25) TOP

Positions the command cursor at node 1 of the first allocated menu block.

(26) BOT

Positions the command cursor at the first free node beyond the last defined command node of the allocated menu block/blocks.

(27) REP,ANAME,NARGS,ADOCODE,ASUBBR,AMNUCODE,AMENU,ADATA

Replaces the command at the current cursor position. The cursor remains unchanged. Arguments to REP are identical to COM.

(28) DEL

Replaces the command at the current cursor position with a null command node. The cursor remains unchanged.

2.13 Examples

For an example of a create mode MENDEL program please see Appendix A. Below are included examples of MENDEL editing, carried out on the MNB file produced by assembling PATH MDL.

Example 1

Replace global command USESTYLUS with a command named USEPEN. The latter has no arguments, is defined by the application procedure XSTY, and is to be executed with the normal delay after command selection. No menu change is to occur and the data area should initially read 'SPARK'.

```
     EDIT,PATH        /OPEN PATH MNB FOR EDITING
     GLOBL            /ENTER CONTEXT 4
     POS,USESTYLUS    /POSITION CURSOR AT USE SYTLUS
     REP,USEPEN,,DO,XSTY,,,SPARK /REPLACE OLD COMMAND
     FIN              /TERMINATE EDIT
```

Example 2

Add a new menu to PATH named BONZO with entry command DOG and exit command BAND. A new global command, MUSIC, will activate menu BONZO. The menu will be saved on menu change.

```
     EDIT,PATH          /OPEN PATH MNB FOR EDITING
     SAVE,TRUE          /ENABLE MENU SAVING
     SBDEC,FOO          /DECLARE A NEW SUBBR
     MNDEC,BONZO        /DECLARE A NEW MENU
     GLOBL              /ENTER CONTEXT 4
     COM,MUSIC,,,,MENU,BONZO /ADD GLOBAL COMMAND
     MENU,BONZO         /BEGIN BONZO MENU DEFINITION
     ENTER              /CURSOR TO COMMAND NODE 1
     COM,DOG,,DO,FOO    /FOO IS ENTRY PROCEDURE
     EXIT               /CURSOR TO COMMAND NODE 2
     COM,BAND,,,,GO,PATH ACTIVATE MENU PATH
     FIN                /TERMINATE EDIT
```

3. USING THE MENDEL EDITOR-ASSEMBLER

The MENDEL editor-assembler may be used either to generate a new MNB binary file from a create mode program or to alter an existing file. The assembler can also produce a binary relocateable Jump Table for PRTE, a source program listing, and a dump of the MNB file.

3.1 Internal Operation of the Editor-Assembler

The MENDEL editor-assembler exists as an execute program consisting of the files MENDEL XCU and MENDEL XCT on the system disc area, <SYS>. When loaded, it first asks the operator for an *option string* consisting of various *control characters* and the application name. According to the parameters received, it may produce, in order, an assembled or edited MNB file, a numbered listing of the source code, a listing of the Menu Address Table, a listing of the Subroutine Name Table, a relocateable Jump Table, and an ASCII dump of the binary MNB file.

3.2 Structure of the Editor-Assembler

The MENDEL execute program is coded in FORTRAN. It is organized so that a single high-level subroutine executes each control option. The main program, named MENDEL, merely retrieves and parses the option string and determines which of the procedures to call.

The actual task of producing an MNB file from source code falls to subroutine YYMDLA. This procedure reads one MENDEL command at a time from the source input device and parses it using borrowed PRTE routines. Each MENDEL statement is interpreted by a single FORTRAN subroutine, entered via a Jump Table, to produce MNB file entries. Global information such as context and error data is contained in common blocks.

Subroutines YYMNUL and YYSBRL list the MAT and SNT of the MNB file on the source listing device. They will not be entered unless YYMDLA has also been called.

The production of the binary relocateable Jump Table is a fairly intricate task executed by the high-level subroutine YYJMPI. This procedure opens the specified MNB file and reads the SNT. YYJMPI outputs a global transfer vector code for each subroutine name declared,along with other necessary loader information. A separate low-level procedure is used to produce each type of code.

When all control options have been processed, MENDEL closes any open files and either exits to DOS or asks for the next option string.

3.3 Loading the Editor-Assembler

Before loading the editor-assembler it may be necessary to make a few device assignments. Table 3-1 describes MENDEL DAT slot usage and recommended devices.

TABLE 3-1

EDITOR-ASSEMBLER DAT SLOT USAGE

OCTAL SLOT	FUNCTION	I/O TYPE	DATA MODE	TYPE OF FILE ACCESS	FILE EXTENSION	RECOMMENDED DEVICE ASSIGNMENTS
3	PROMPTING MESSAGES	OUTPUT	5/7 ASCII	NONE	NONE	TTA
	OPERATOR COMMANDS	INPUT	5/7 ASCII	NONE	NONE	TTA
	ERROR MESSAGES	OUTPUT	5/7 ASCII	NONE	NONE	TTA
13	SOURCE LISTING	OUTPUT	5/7 ASCII	SEQUENTIAL	LST	DKA,TTA
	ERROR MESSAGES	OUTPUT	5/7 ASCII	SEQUENTIAL	LST	DKA,TTA
	MAT LISTING	OUTPUT	5/7 ASCII	SEQUENTIAL	LST	DKA,TTA
	SNT LISTING	OUTPUT	5/7 ASCII	SEQUENTIAL	LST	DKA,TTA
	DUMP LISTING	OUTPUT	5/7 ASCII	SEQUENTIAL	DMP	DKA,TTA
	JUMP TABLE	OUTPUT	BIN	SEQUENTIAL	BIN	DKA*
17	MNB FILE	OUTPUT	BIN	RANDOM	MNB	DKA
	MNB FILE (FOR MAT, SNT, DUMP)	INPUT	BIN	RANDOM	MNB	DKA
20	SOURCE COMMANDS	INPUT	5/7 ASCII	SEQUENTIAL	MDL	DKA,TTA

*Slot 13 may not be assigned to TTA if a Jump Table is to be output.

Note that if slot 13 is assigned to TTA, the Jump Table control character may not be included in the command string. VT may be ON or OFF. If slots 13, 14, and 20 are all file-oriented, 4 I/O buffers must be allocated using the DOS BUFFS command. DAT slot 20 must be assigned to TTA if the 'I' option is to be used. Since MENDEL XCT and XCU are located on the system disc area, DAT slot -4 should be assigned to <SYS>.

To load the MENDEL editor-assembler, type:

```
$ E MENDEL
```

The example below illustrates the loading procedure necessary to assemble the create mode MENDEL program, PATH MDL.

```
$ DOS V2A
$ A <SYS> -4 <cr>   /LOAD FROM SYSTEM DISC
$ A DK 2Ø  <cr>     /PATH MDL IS DISC FILE
$ BUFFS 4  <cr>     /NEED 4 I/O BUFFERS
$ E MENDEL <cr>     /LOAD EDITOR-ASSEMBLER
```

3.4 Operator Commands to the Editor-Assembler

When the editor-assembler is loaded and running it will issue the prompting message:

```
MENDEL
OPT ← FNAME?
>
```

The character > is an invitation to type an option string. The latter consists of a string of control characters followed by the character ← and the application name.

Each file accessed by the editor-assembler will have the same first name as the application name supplied in the option string. A different extension is used for the various I/O files, however. For example, when assembling the MENDEL program PATH, the following files may be referenced:

PATH MDL	SOURCE FILE
PATH MNB	BINARY OBJECT (FOR PRTE)
PATH LST	SOURCE, MAT, SNT LISTING
PATH BIN	JUMP TABLE (LOADABLE)
PATH DMP	DUMP FILE

The source, MAT, and SNT listings will be included one after another in file PATH LST if slot 13 is assigned to a file-oriented device.

The control character part of the option string determines which editor-assembler functions will occur. Multiple control characters should be concatenated without commas. A list of the characters and their functions is given in TABLE 3-2. Although control characters may appear in any order in the option string, the functions they select always occur in the order given in the table. Thus if the character E is included, MENDEL will ignore all other control characters and immediately exit to DOS.

TABLE 3-2

EDITOR-ASSEMBLER CONTROL CHARACTERS

COMMAND FORMS: <CONTROL CHARACTERS>+<FILENAME><*cr*>

E <*cr*>

CHARACTER	EXECUTION ORDER	FUNCTION
E+	1	Exit to DOS immediately
B	2	Execute the MENDEL statements in file <FILENAME> MDL to create or edit file <FILENAME> MNB. DAT slot 20 should be file oriented. Syntax errors abort processing.
I	3	Interactively execute the MENDEL statements input via DAT slot 20 to create or edit file <FILENAME> MNB. Slot 20 must be assigned to TTA. Commands may be retyped if syntax errors occur. MENDEL command FIN causes termination
L	4	List source code on file <FILENAME> LST(only if B or I option also used).
M*	5	List Menu Address Table on file <FILENAME> LST(only if B or I option also used).
S*	6	List Subroutine Name Table on file <FILENAME>(LST only if B or I option also used).
J	7	Produce a Jump Table for file <FILENAME> MNB in file <FILENAME> BIN.
O	8	Produce a DUMP of MNB file <FILENAME> MNB in file <FILENAME> DMP.
A	9	Do all of the above except E and I.

+ May only be used by itself.
* Ignored if L not also included.
  If B and I are both included, B is ignored.

Typing the option string below will cause MENDEL to assemble the create mode program contained in file PATH MDL and produce a Jump Table in file PATH BIN.

```
MENDEL
OPT←FNAME?
>BJ←PATH<cr>
```

3.5 List File Format

If the L control character is included in the option string, each MENDEL statement or comment line encountered will be numbered and output to the list file device. Unfortunately the character <tab> is not recognized by some of the handlers and so will not be output.

If the M option is selected, a listing of the MAT will follow the source listing. The following format is used:

MENU NAME   BLOCK NUMBER IN MNB FILE

If the S option is included in the option string, an SNT listing will follow the MAT printout. The application subroutine names are simply listed in the order they occur in the SNT and Jump Table.

3.6 Dump File Format

The Dump facility of MENDEL is necessary because the MNB file is output in IOPS BINARY data mode and cannot be inspected using the DOS text editor. The layout of a DMP file closely follows the MNB file structure given in Appendix E.

Dump files are broken down into sections according to MENDEL contexts and physical blocks (256 words). The following labels identify contexts 1-6:

Label	Context
HEADER	1
MENU TABLE	2
SUBROUTINE TABLE	3
GLOBALS	4
ARGET	5
MENUS	6

New blocks are identified by the offset label:

```
***** BLOCK n
```

The entries comprising each context are numbered in order of occurrence and listed in convenient formats.

Header block entries are word-numbered and listed as either integer values or text strings. The meaning of each entry should be clear from Appendix C.

The MAT and SNT entries appear exactly as they would in a list file produced by the M and S control characters,

The portions of the DUMP file describing the GLOBAL, ARGET, and MENU contexts use a common format for listing menu blocks. A word-numbered list of the header node values is output followed by a decoded entry for each of the 24 command nodes. Table 3-3 describes the layout and meaning of each field of a command node listing.

3.7 Prompting Messages

MENDEL types the message:

```
MENDEL
OPT←FNAME?
>
```

to indicate its readiness to accept an option string. If the I control character is included, > is also the invitation to type the next MENDEL statement.

Although MNB files may be edited using statements in an MDL file, they are normally altered interactively using the I control character. As an aid, edit mode MENDEL statements which reference the command cursor always cause the altered command node to be decoded and listed on DAT slot 3 (TTA). The listing format is identical to that given in Table 3-3 for Dump files - minus the labels. The POS statement is particularly useful for examining MNB file command nodes.

3.8 Error Messages

All error messages output by the editor-assembler, except one, go to both the source list file and DAT slot 3. The message:

```
COMMAND ERROR, IGNORED
```

is output only to the teletypes if an illegal control character or poorly-formed option string has been entered. The malformed string is ignored and the corrected version should be retyped.

As each MENDEL statement is encountered, the editor-assembler checks its syntax and the validity of its arguments. The action taken by MENDEL if an error occurs depends upon whether or not the I control character was included in the option string.

If it was not included, MENDEL lists the offending statment and outputs the following messages to the source list device and TTA:

K   *STATEMENT CAUSING ERROR*
ERROR NUMBER *N*
***** SUCCEEDING COMMANDS NOT OBEYED *****

where *K* is the line number and *N* is the error number. All files are closed and succeeding commands are not executed but are checked for superficial syntax errors. Assembly or editing fails and subsequent control characters are ignored. The MNB file may no longer exist or it may be garbage.

TABLE 3-3

COMMAND NODE LISTING FORMAT

LABEL	*SMALL L. BUTTONS*	IDX	COMMAND	DATA	ARG	I	W	SUB	EXE		MENU
	LARGE L. BUTTONS	IDX	COM	DATA	ARG	I	W	SUB	EXE		MENU
DATA TYPE		INTEGER	TEXT	TEXT	INTEGER	INT.	INT.	INT	INT.	INT.	INTEGER
FUNCTION		Starting word index (not cursor index) of command node within a menu block.	5 or 9-character command name. Blank if NULL command.	5 or 9-character data area. If command is unaccessed, the characters NULL will appear here.	Number of arguments.	1 if command is inactive (DONT,DTNOW)	1 if command is to be executed with no delay (DONOW,DTNOW).	Index of associated application procedure in the SNT. ∅ means none associated.	1 if exit procedure of old menu is to be executed on menu change.	1 if entry procedure of new menu is to be executed on menu change.	Block number of new menu to be activated. ∅ means no menu change will occur.

If the I control character was included in the option string and a fault occurs, only the error number will be typed out. The offending statement is ignored and MENDEL asks for it to be retyped by outputting the prompting character, >.

Upon termination of an edit or assembly, MENDEL outputs several messages to both TTA and the list device. The first of these indicates the number of errors and source lines encountered:

```
    ********** n ERRORS, m LINES.
```

The second message notes either the success or failure of MENDEL processing:

```
    NORMAL EXIT  or

    ********** ASSEMBLY ABORTED AT LINE k
```

Numbered MENDEL errors are explained in Appendix K along with PRTE diagnostics. Typing *<cntrl P>* will *not* restart the editor-assembler.

3.9 Examples

The two examples below illustrate the complete loading procedure and option strings necessary to assemble file PATH MDL and edit PATH MNB.

Example 1

Assemble PATH MDL and produce a source, MAT, SNT, and Dump listing on TTA.

```
    $ DOS V2A
    $ A <SYS> -4 <cr>       /LOAD FROM SYSTEM DISC
    $ A DK 2Ø <cr>          /PATH MDL IS DISC FILE
    $ A TTA 13 <cr>         /LISTING TO TTA
    $ VT ON <cr>
    $ <cntrl X>             /TURN ON DISPLAY
    $ E MENDEL <cr>         /ONLY NEED 3 BUFFERS
    MENDEL
    OPT←FNAME?              /OPTION STRING?
    >BLMSD←PATH <cr>        /NO JUMP TABLE ALLOWED
                                .
                                .
                                .
                  SOURCE LISTING OF PATH
                                .
                                .
                                .

    ********** Ø ERRORS, N LINES. /END OF ASSEMBLY
    NORMAL EXIT
                                .
                                .
                                .
                       MAT LISTING
                       SNT LISTING
                       DUMP LISTING
                                .
                                .
                                .

    MENDEL
    OPT←FNAME?              /NEXT OPTION STRING?
   > E <cr>                 /EXIT TO DOS.
   $ DOS V2A
   $
```

Example 2

Edit PATH MNB, making the lightpen the starting stylus device.

```
     $ DOS V2A
     $ A <SYS> -4 <cr> /LOAD FROM SYSTEM DISC
     $ A TTA 20 <cr>   /INTERACTIVE COMMAND INPUT
     $ E MENDEL <cr>   /ONLY NEED 3 BUFFS
     MENDEL
     OPT←FNAME?        /OPTION STRING?
     >I←PATH <cr>      /INTERACT
     >EDIT,PATH <cr>   /EDIT MODE
     >STYLUS,LPN <cr>  /BAD COMMAND NAME
     ERROR NUMBER 8    /COMMAND IGNORED
     >STYLS,LPN <cr>   /USE LIGHTPEN
     >FIN <cr>         /END INTERACTION
     NORMAL EXIT       /NO LINE COUNT GIVEN
     MENDEL
     OPT←FNAME?        /NEXT OPTION STRING
    >E <cr>            /EXIT TO DOS
     $ DOS V2A
     $
```

4. OPERATION OF THE PIGS RUN TIME ENVIRONMENT

The PIGS run time environment is coded largely in FORTRAN IV. The routines may be grouped by function into four sets: initialisation, command polling, command reconciliation, and menu activation. Figures 4-1 through 4-5 include flowcharts of these four functions and the PRTE main control loop. Each function is flowcharted separately and linked to the main control loop by a circled letter. A general discussion of the run time environment is included below.

4.1 PRTE - Main Control Loop

When a Graphics Application Program is loaded using EXECUTE, PRTE subroutine PIGS receives control via a JMS* instruction in the Jump Table. This JMS* instruction is output by the MENDEL editor-assembler and serves merely to ease the overlay construction process using XCHAIN.

4.2 PRTE - Initialisation

Subroutine PIGS controls all PRTE functions. It first initialises the run time environment, interpreting header information contained in block 1 of the MNB file output by MENDEL. Then the menu blocks containing global command nodes are read into the bottom of the Command Table array. The size of the common block containing the Command Table is adjusted at load time by a pseudo-instruction located in the Jump Table so that all global command nodes will fit into core. FOG subroutines are used to generate the display file structure (see Appendix D) which, when executed by the VT15 processor, creates the CRT display. Any error which occurs during PRTE initialisation will cause the program to terminate with a teletype message.

Having initialised the display and run time environment, subroutine PIGS activates the starting menu of the application and executes its entry procedure. A new menu is activated simply by reading the appropriate MNB block into the Command Table above the globals. The lightbutton display files need not be changed since the text command names are referenced indirectly as addresses in the Command Table. It is only necessary to blank off the lightbutton display files during menu block input.

For the remainder of an interaction session, PRTE simply remains in the following command interpretation loop:

(1) Poll active devices until a command is selected.

(2) Retrieve the command's name and arguments.

(3) Execute the associated application procedure, if required.

(4) Retrieve a new menu, if required, and go to (1).

The steps are briefly discussed below.

4.3 PRTE - Command Polling

Interruptions such as error messages or typing *<cntrl* P> cause control to be transferred to step (1), above. In polling for a new command, PRTE repeatedly examines each active device until the operator selects a command or until a clock scheduled command becomes due. Polling immediately ceases and command selection information is returned to subroutine PIGS.

4.4 PRTE - Command Reconciliation and Execution

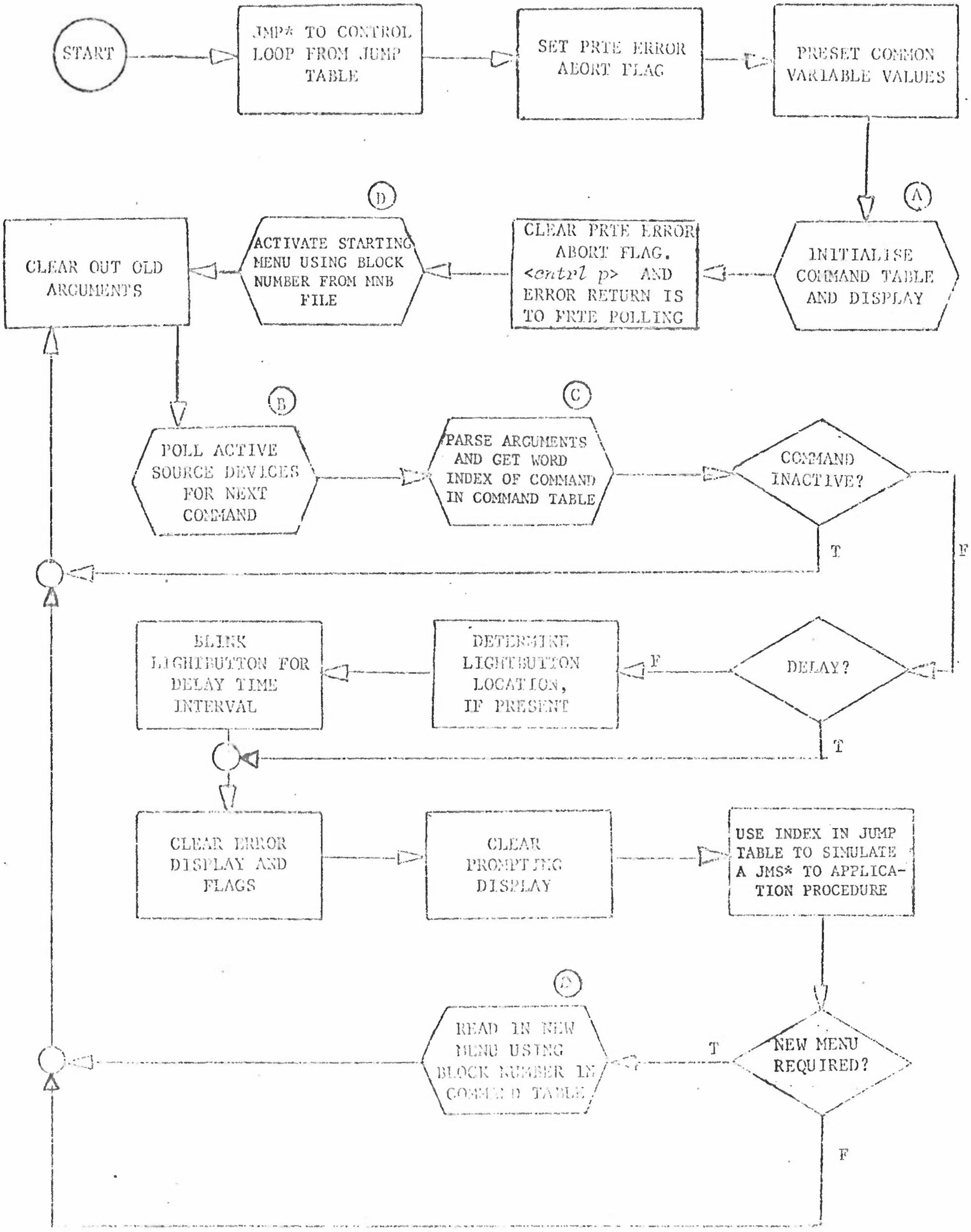
Because the form of the command information varies from device to device, PRTE must next convert it into its standard internal format: a text command name and the word index of the related Command Table node. This reconciliation process includes argument parsing in the case of typed input.

At this point subroutine PIGS blinks the associated lightbutton, if it exists, and clears the previous error and prompting messages. If an application procedure is associated with the command, the index of its starting address in the Jump Table is found in the command node. A JMS* to the application subroutine is simulated. Control returns directly to PIGS.

4.5 PRTE - Menu Activation

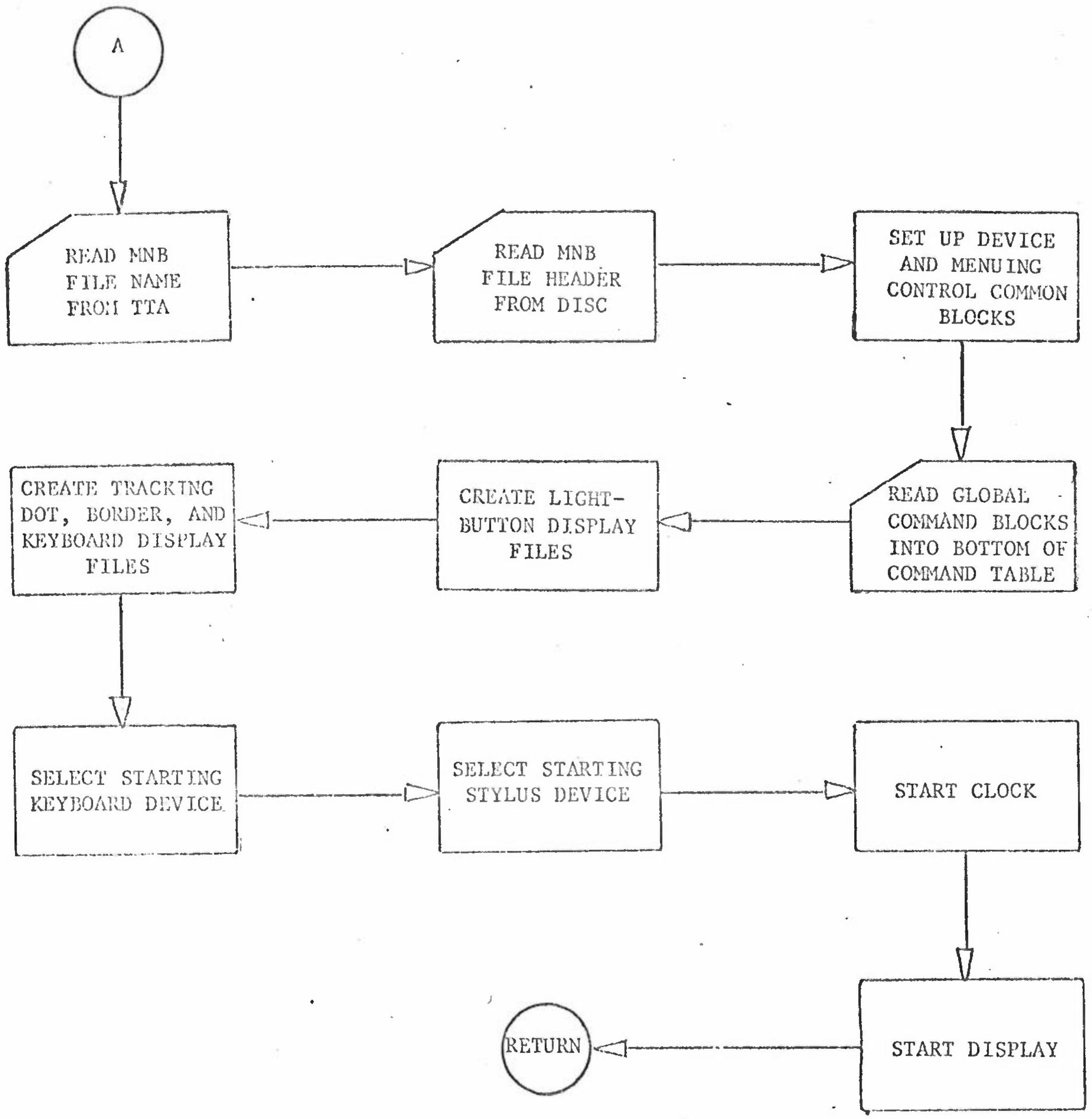
As the final step in the command interpretation loop, PRTE examines the selected command node after command execution to determine whether a new menu must be activated. If so, the setting of two bits in the node determines whether or not the exit and entry routines for the old and new menus, respectively, will be executed. Activation of a new menu occurs as described in Section 4.2.

FIGURE 4-1 PRTE - MAIN CONTROL


START
JMP* TO CONTROL LOOP FROM JUMP TABLE
SET PRTE ERROR ABORT FLAG
PRESET COMMON VARIABLE VALUES
A
INITIALISE COMMAND TABLE AND DISPLAY
CLEAR PRTE ERROR ABORT FLAG. <cntrl p> AND ERROR RETURN IS TO PRTE POLLING
D
ACTIVATE STARTING MENU USING BLOCK NUMBER FROM MNB FILE
CLEAR OUT OLD ARGUMENTS
B
POLL ACTIVE SOURCE DEVICES FOR NEXT COMMAND
C
PARSE ARGUMENTS AND GET WORD INDEX OF COMMAND IN COMMAND TABLE
COMMAND INACTIVE?
T
F
DELAY?
F
DETERMINE LIGHTBUTTON LOCATION, IF PRESENT
BLINK LIGHTBUTTON FOR DELAY TIME INTERVAL
T
CLEAR ERROR DISPLAY AND FLAGS
CLEAR PROMPTING DISPLAY
USE INDEX IN JUMP TABLE TO SIMULATE A JMS* TO APPLICATION PROCEDURE
D
READ IN NEW MENU USING BLOCK NUMBER IN COMMAND TABLE
T
NEW MENU REQUIRED?
F

NOTE: Circled letters correspond to flowcharts on succeeding pages.

FIGURE 4-2 PRTE - COMMAND TABLE AND DISPLAY INITIALISATION



A
READ MNB FILE NAME FROM TTA
READ MNB FILE HEADER FROM DISC
SET UP DEVICE AND MENUING CONTROL COMMON BLOCKS
READ GLOBAL COMMAND BLOCKS INTO BOTTOM OF COMMAND TABLE
CREATE LIGHT-BUTTON DISPLAY FILES
CREATE TRACKING DOT, BORDER, AND KEYBOARD DISPLAY FILES
SELECT STARTING KEYBOARD DEVICE
SELECT STARTING STYLUS DEVICE
START CLOCK
START DISPLAY
RETURN

FIGURE 4-3 PRTE - COMMAND POLLING

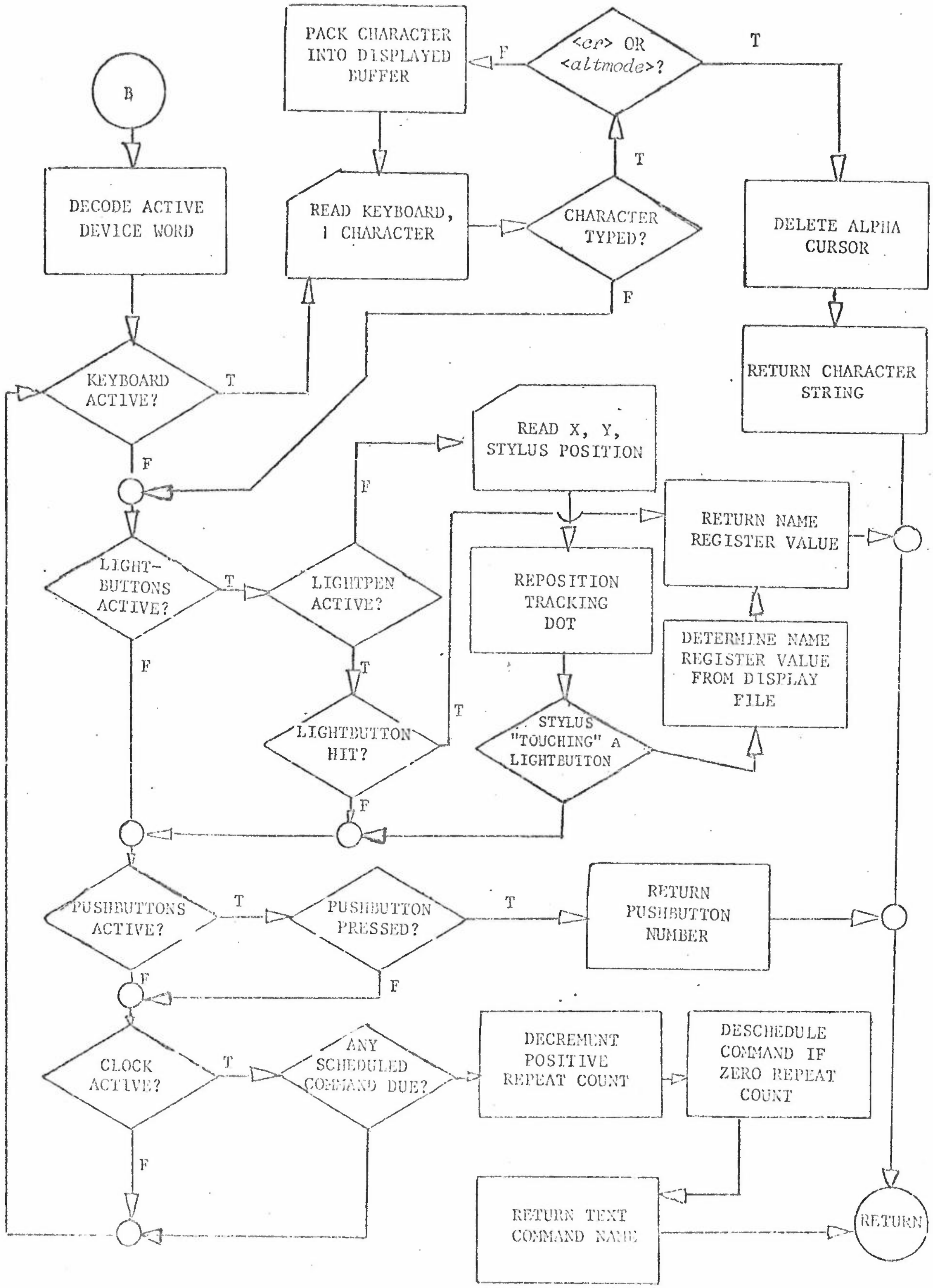

PACK CHARACTER INTO DISPLAYED BUFFER
F
<cr> OR <altmode>?
T
B
T
DECODE ACTIVE DEVICE WORD
READ KEYBOARD, 1 CHARACTER
CHARACTER TYPED?
DELETE ALPHA CURSOR
F
RETURN CHARACTER STRING
KEYBOARD ACTIVE?
T
F
READ X, Y, STYLUS POSITION
F
RETURN NAME REGISTER VALUE
LIGHT-BUTTONS ACTIVE?
T
LIGHTPEN ACTIVE?
REPOSITION TRACKING DOT
DETERMINE NAME REGISTER VALUE FROM DISPLAY FILE
F
T
T
LIGHTBUTTON HIT?
STYLUS "TOUCHING" A LIGHTBUTTON
F
PUSHBUTTONS ACTIVE?
T
PUSHBUTTON PRESSED?
T
RETURN PUSHBUTTON NUMBER
F
F
CLOCK ACTIVE?
T
ANY SCHEDULED COMMAND DUE?
DECREMENT POSITIVE REPEAT COUNT
DESCHEDULE COMMAND IF ZERO REPEAT COUNT
F
RETURN TEXT COMMAND NAME
RETURN

FIGURE 4-4 PRTE - COMMAND RECONCILIATION

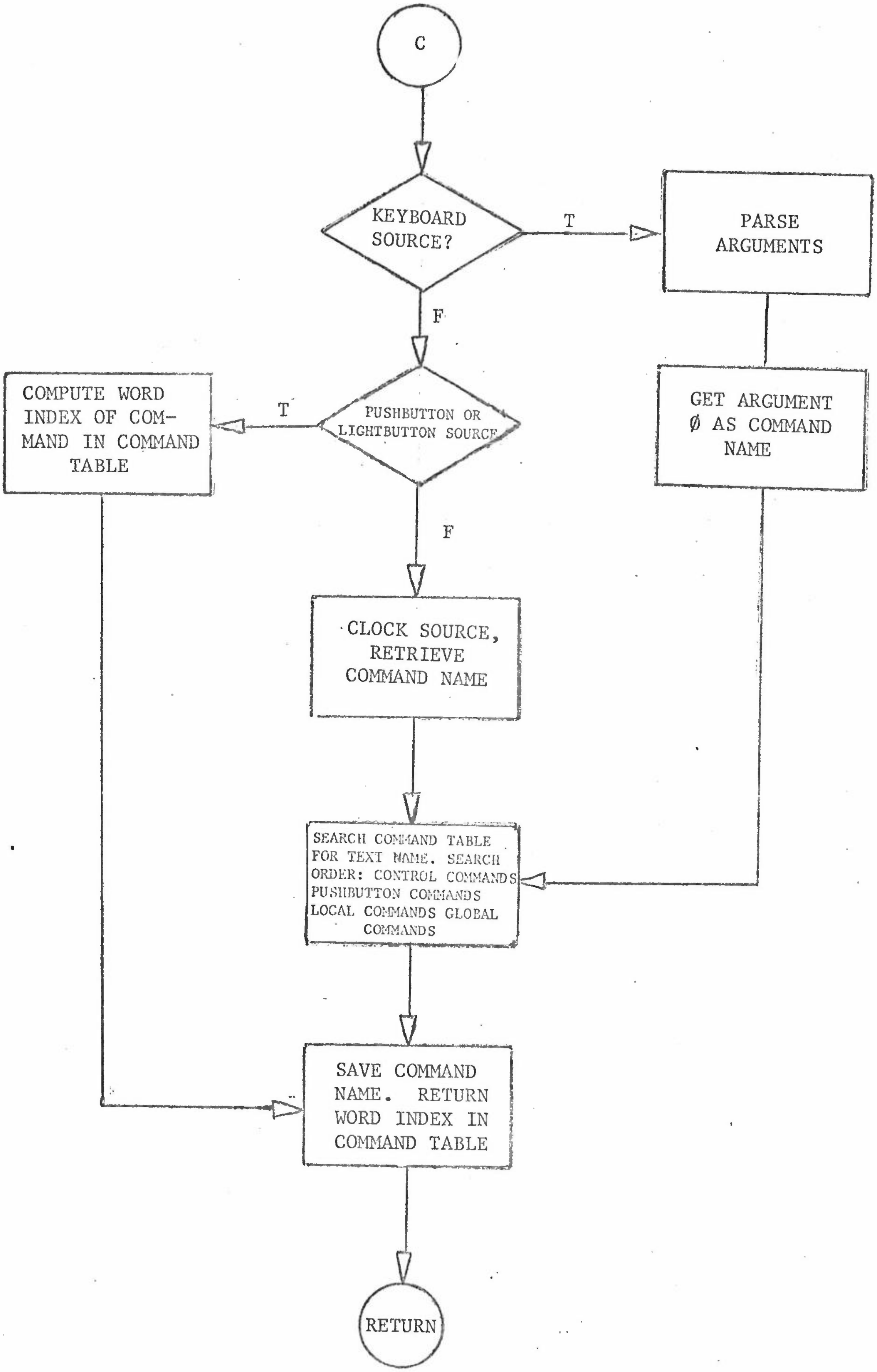
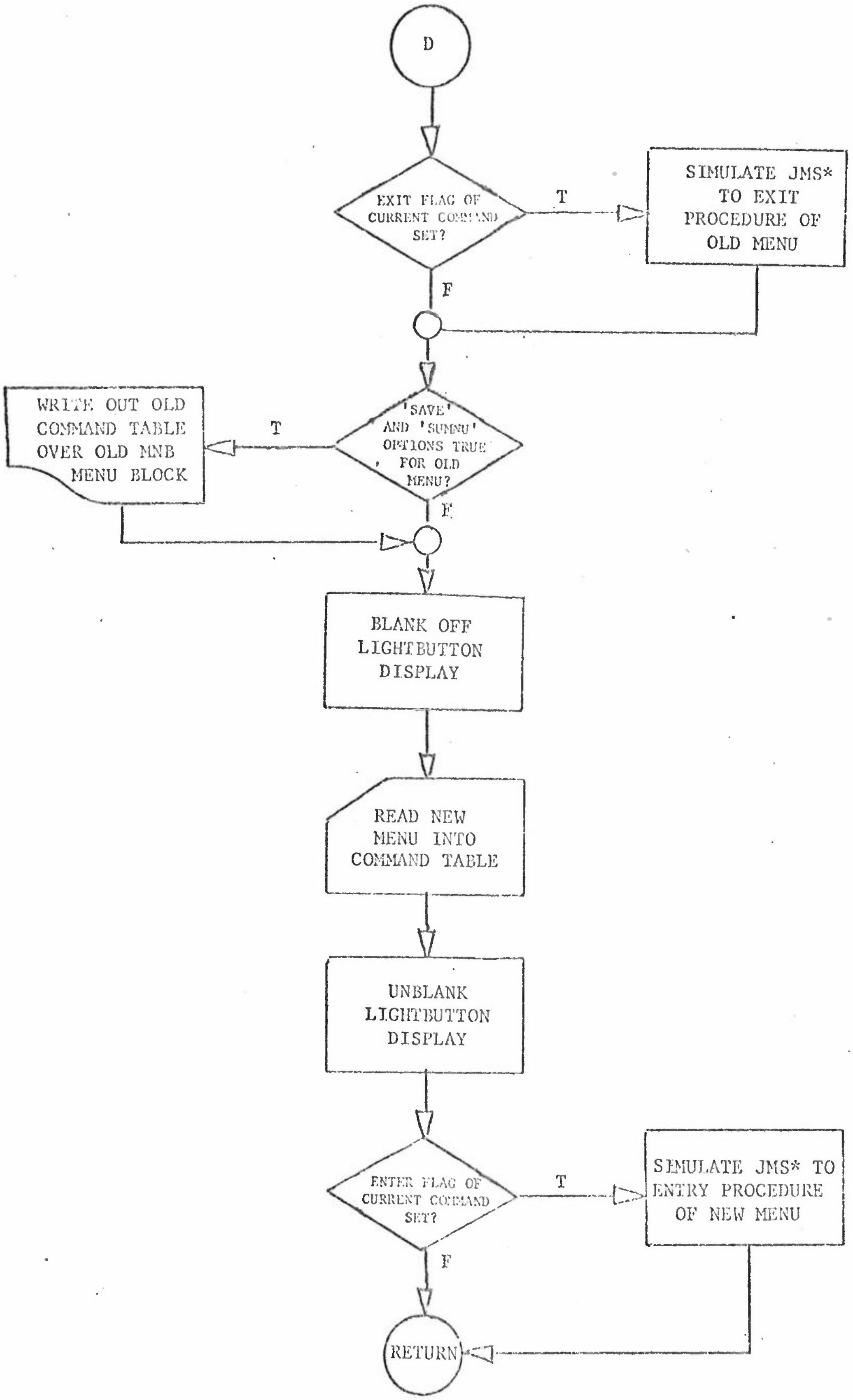

C
KEYBOARD SOURCE?
T
PARSE ARGUMENTS
F
COMPUTE WORD INDEX OF COMMAND IN COMMAND TABLE
T
PUSHBUTTON OR LIGHTBUTTON SOURCE
GET ARGUMENT Ø AS COMMAND NAME
F
CLOCK SOURCE, RETRIEVE COMMAND NAME
SEARCH COMMAND TABLE FOR TEXT NAME. SEARCH ORDER: CONTROL COMMANDS PUSHBUTTON COMMANDS LOCAL COMMANDS GLOBAL COMMANDS
SAVE COMMAND NAME. RETURN WORD INDEX IN COMMAND TABLE
RETURN

FIGURE 4-5 PRTE - MENU ACTIVATION


D
EXIT FLAG OF CURRENT COMMAND SET?
T
SIMULATE JMS* TO EXIT PROCEDURE OF OLD MENU
F
'SAVE' AND 'SUMNU' OPTIONS TRUE FOR OLD MENU?
T
WRITE OUT OLD COMMAND TABLE OVER OLD MNB MENU BLOCK
F
BLANK OFF LIGHTBUTTON DISPLAY
READ NEW MENU INTO COMMAND TABLE
UNBLANK LIGHTBUTTON DISPLAY
ENTER FLAG OF CURRENT COMMAND SET?
T
SIMULATE JMS* TO ENTRY PROCEDURE OF NEW MENU
F
RETURN

5. WRITING A GRAPHICS APPLICATION PROGRAM

There are essentially six tasks in developing a graphics application using PIGS, most of which are common to the design of any large program. These are listed below along with pertinent references:

(1) Determine the desired commands and menus and describe them to PIGS by writing a MENDEL program (Chapters 2 and 3).

(2) Design the application display (FOG manual).

(3) Design the application data base.

(4) Write an application procedure in FORTRAN IV or MACRO-15 assembly language for each command (Chapter 5, FORTRAN and MACRO-15 manuals).

(5) Debug the application procedures.

(6) Determine the overlay structure of the application procedures and data base. Combine this overlay structure with that of PRTE using XCHAIN (Chapter 6, CHAIN manual, PDP15 User Notes 2 and 3).

This chapter deals largely with task (4), writing application procedures. Examples from PATH are included in Section 5.4.

5.1 Initialising a Graphics Application Program

A graphics application program running under PIGS ordinarily consists of some data tables or common blocks, a display file structure, and a set of application procedures. When PRTE begins running, the entry procedure of the starting menu will be automatically executed. This procedure should initialise the application data base and may be used to direct preliminary operator choices by activation and deactivation of displayed commands.

The entry procedure of the starting menu may also define the application display using FOG. To link the main application display file to the PIGS display, use the FOG commands:

```
CALL RCHOOS (16)
CALL DRAW (1,MAIN(1))
```

after defining the main display file, MAIN. The FOG commands:

```
SCHOOS (16), DINIT,RINIT, and SINIT
```

should *never* be used by the application program. Remember that the display is active. All files must be well-formed before the main file is linked to the PRTE display. The initial display attributes set by FOG are unchanged by the PRTE display.

5.2 Argument Transmission in Application Procedures

The principal difference between ordinary FORTRAN IV or MACRO-15 routines and graphics application procedures written using these languages is in the method of retrieving arguments parsed by the run time environment. To allow more flexibility in the number and types of arguments a command may have, PRTE classifies and places arguments in several common blocks accessible to both the run time environment and the GAP. Actual argument values are retrieved by application procedures using special PRTE functions. Application procedures are coded as subroutines with no *formal* arguments.

Each PRTE argument-getting routine is a logical function which is .TRUE. if the argument was specified and .FALSE. if it was omitted. Each of these functions has at least 2 parameters: an argument index and an argument variable to hold the returned value. The index is simply the position of the desired argument in a complete command string, numbered from left to right with argument number 1 being the text command name. If the desired argument was omitted, the value of the supplied argument variable remains unchanged, allowing easy specification of default values.

Table 5-1 describes the various PRTE argument-getting functions and their parameters. Although the functions correspond to particular FORTRAN data types, they may also be used from MACRO-15 procedures by utilising the FORTRAN argument transmission protocol (see Chapter 3, PDP15 FORTRAN IV OPERATING ENVIRONMENT manual).

Occasionally it is useful for one application procedure to call another. In this case a second set of PRTE routines may be used to put arguments in the argument common block for retrieval by the called procedure. These *argument-putting* subroutines are described in Table 5-2. Before each call of an application procedure, PRTE argument common must be initialised. Arguments may then be put in the common area and the application procedure called. For convenience, PRTE initialises the argument common block when the argument index of any argument-putting routine used is negated. The example below illustrates the FORTRAN equivalent of the command:

```
    SHOW,3<cr>
```

in PATH.

```
    CALL PUTARG (-2,3)
    CALL SHOW
```

Further information about a command and its arguments may be obtained by a GAP directly from the argument common blocks manipulated by PRTE. Of particular use are argument types, command source device, and number of arguments specified. Argument common blocks ARGTP, CARG, and DARG are described in Appendix C.

5.3 Special PRTE Subroutines

Although MENDEL may be used to set up the initial state of PRTE active devices, commands, data areas, and display, it cannot control PRTE flexibly during interaction. Instead, FORTRAN-callable subroutines have been provided to enable dynamic control over selected PRTE functions (such as data area display and command scheduling). Table 5-3 describes all currently available PRTE special subroutines. These subroutines receive their arguments via the normal FORTRAN IV calling sequence.

TABLE 5-1

PRTE ARGUMENT - GETTING FUNCTIONS*

SUBROUTINE NAME	ARGUMENTS	FILE NAME	DESCRIPTION	GLOBAL REFERENCES
GETCH		GETCH2	Get character string, left justified, blank filled.	ABORT
	NOARG		Integer index of argument	MVC
	STRING		First element of real array to hold returned string	
	NOCHAR		Returns number of characters in string argument	
	MAXCH		Maximum number of characters before overflow	
GETDI		GETDI2	Get double integer number	GETDP
	NOARG		Integer index of argument	
	DUBINT		Returns double integer argument value	
GETDP		GETDP2	Get double precision number	ABORT
	NOARG		Integer index of argument	
	DUBPRE		Returns double precision argument value	
GETFIL		GETFIL	Get text file name and extension. If extension is	GETCH
	NOARG		present, it must be separated from the file name by	FNAME
			a blank (use quotes)	
	STRING		First element of real 2-word array to return file	
			name and extension	
	EXT		Left justified, 3-character default extension	
GETLOG		GETLOG	Get logical argument. Either the string TRUE or	
			FALSE must have been typed.	
	NOARG		Integer index of argument	ABORT
	LOG		Returns logical value, .TRUE. or .FALSE.	GETCH
GETSI		GETSI2	Get single integer number	ABORT
	NOARG		Integer index of number	GETDP
	INTEGR		Returns integer argument value	
GETSR		GETSR2	Get single real number	GETDP
	NOARG		Integer index of argument	
	REALNO		Returns real argument value	

*Each logical function may also be used as a subroutine.
Functions are .FALSE. if the desired argument was omitted.

TABLE 5-2

PRTE ARGUMENT-PUTTING SUBROUTINES

SUBROUTINE NAME	ARGUMENTS	FILE NAME	DESCRIPTION	GLOBAL REFERENCES
PUTCH		PUTCH2	Puts a character string in argument common	ABORT
	NOARG*		Integer argument index	MVC
	STRING		First element of real array containing string argument	
	NOCHAR		Number of characters in string to be moved	
PUTDI		PUTDI2	Puts a double integer number in argument common	PUTDP
	NOARG*		Integer argument index	
	DUBINT		Double integer number to be moved	
PUTDP		PUTDP2	Puts a double precision number in argument common	ABORT
	NOARG*		Integer argument index	
	DUBPRE		Double precision number to be moved	
PUTSI		PUSI2	Puts a single integer number in argument common	PUTDP
	NOARG*		Integer argument index	
	INTEGR		Integer number to be moved	
PUTSR		PUTSR2	Puts a real number in argument common	PUTDP
	NOARG*		Integer argument index	
	REALNO		Real number to be moved	

*First argument index used before the application procedure call must be negated to initialise PRTE argument common blocks.

TABLE 5-3

SPECIAL PRTE SUBROUTINES

SUBROUTINE. NAME	ARGUMENTS	FILE NAME	DESCRIPTION	GLOBAL REFERENCES
ACTCM		ACTCM	Activates a command in the current menu. Ignored if command already active	CMFIND INBITS MVC
	COMNAM		First element of array containing 5/7 ASCII command name	
CLERR		CLERR2	Clears the current error message display	MVC
CLPRM		CLPRM	Clears the current prompting message display	MVC
DACTCM		DACTCM	Deactivates a command in the current menu. Ignored if command already inactive	CMFIND INBITS MVC
	COMNAM		First element of array containing 5/7 ASCII command name	
DSKEDU		DSKEDU	Deschedules a command in the PRTE clock schedule buffer. No action if command not scheduled	DSCHED
	COMNAM		First array element of the 9-character command name to be descheduled	
DISERR		DISERR	Displays a message in the error display area and rings the teletype bell	PRIN1 MESDIS
	STRING		First array element of the 5/7 ASCII message to be displayed	
	NOCHAR		Number of characters in the message	
	IREST		*<cntrl* P> restart address. (Use NRMRET in common block ERRCON if no special address is desired.)	
ERP		ERP	Displays a GAP error message number and returns to calling procedure	ERRMES
	NUMBER		This integer value +200 will be displayed as the error number	
PROMPT		PROMPT	Displays a message in the prompting message display area	MESDIS
	STRING		First array element of the 5/7 ASCII message to be displayed	
	NOCHAR		Number of characters in the message	

TABLE 5-3 (Continued)

SPECIAL PRTE SUBROUTINES

SUBROUTINE NAME	ARGUMENTS	FILE NAME	DESCRIPTION	GLOBAL REFERENCES
QUIT		QUIT	Exits PRTE and returns to DOS	
SKEDUL		SKEDUL	Schedules a command in the PRTE clock schedule buffer. The command will be selected when the time interval specified has elapsed	SCHED
	COMNAM		First array element of the 9-character command name	
	INTSEC		Interval in seconds before the scheduled command becomes due again	
	INTPLS		Interval in clock pulses (20 milliseconds) before the scheduled command becomes due again. Total interval is INTSEC + INTPLS	
	NRPEAT		Number of times the command is to be executed NRPEAT = -1 means execute indefinitely	
WDAT		WDAT	Displays a 5 or 9-character message in a specified command data area. The number of characters used is dependent upon the display size	WMNU
	COMNAM		First array element of the 9-character command name whose data area is to be changed	
	STRING		First array element of the 5 or 9-character 5/7 ASCII string to be displayed	

5.4 Examples

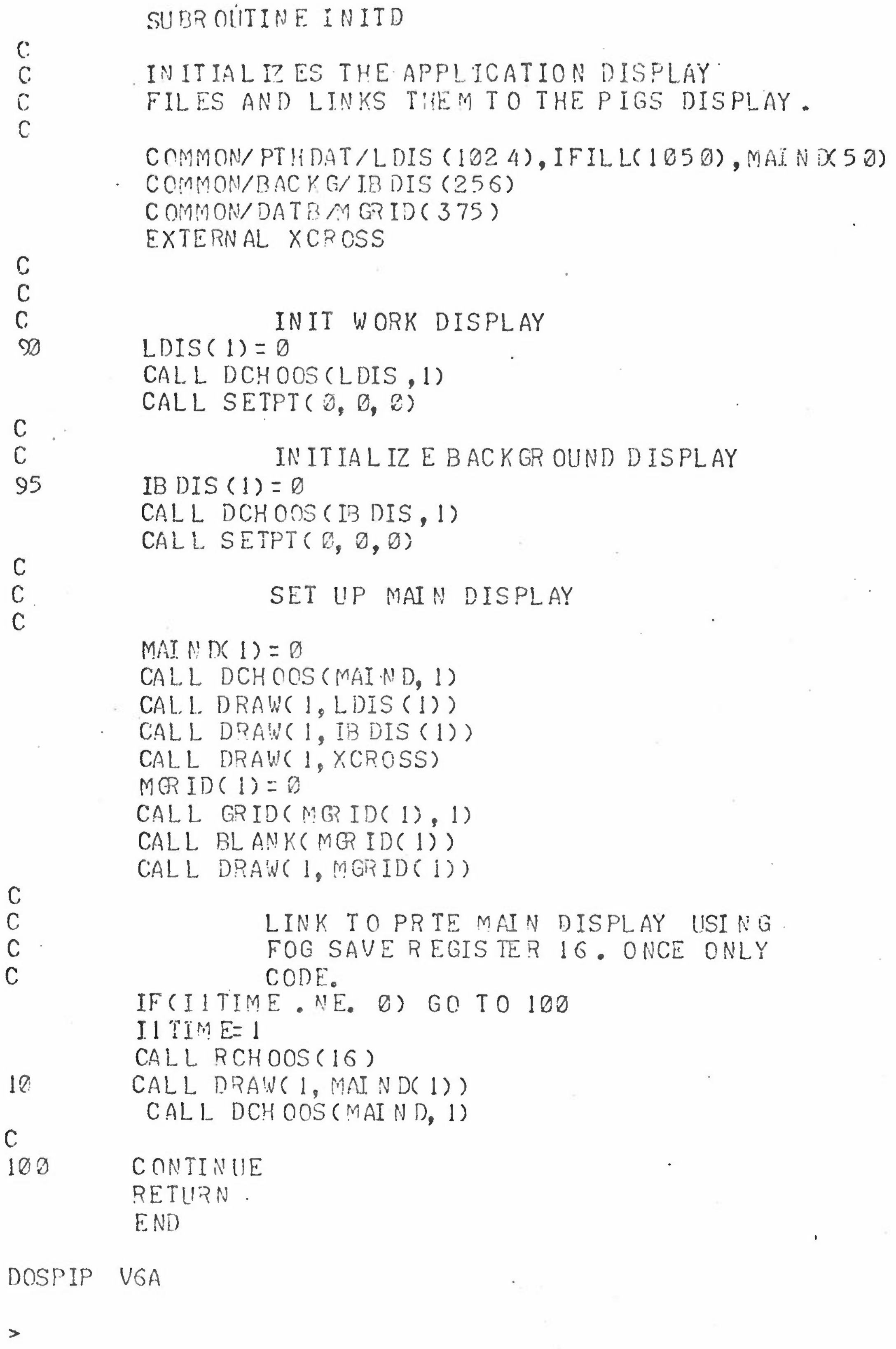
Below are included several examples of GAP application procedures extracted from PATH. The first subroutine, is called by PINIT, the entry procedure for the starting menu of PATH. Note that it creates the PATH display file structure and links it to the PRTE display using FOG save register 16. As INITD may also be entered via command PATH in menu PATH, care has been taken to see that the display linking code is executed once only (INITD may not be overlayed, however, since I1TIME is not in common).

The second example illustrates the use of the PRTE argument-getting and data area display routines. Subroutine SHOWIT is entered upon selection of command SHOW with any active source device. The subroutine behaves slightly differently if the source device was not a keyboard (IDCOME ≠ 1): the current cel number, NUM, is incremented and becomes the default argument value (all arguments are considered to be of type omitted when the source device is not a keyboard). The subsequent subroutine call to GETSI either retrieves a typed integer value or passes on the default value of NUM. In order to display the cel or path number in the data area of command SHOW, it was first necessary to use the FORTRAN ENCODE statement to convert the integer to a 5/7 ASCII string. To use PRTE subroutine WDAT to display the data, it is necessary to provide the text command name, CNAM(1), as an argument. The coding used may cause a problem if one later decides to change the command name using MENDEL, and then forgets to change the data statement in the application procedure. This annoyance may be avoided by retrieving the command name using the GETCH function:

```
     CALL GETCH(1,CNAM(1),NOCHAR,9)
     CALL WDAT(CNAM(1),CDAT(1))
```

The above coding causes the same data area display and makes procedure SHOWIT more independent of the MENDEL command specification.

Example 1



```
          SUBROUTINE INITD
C
C         INITIALIZES THE APPLICATION DISPLAY
C         FILES AND LINKS THEM TO THE PIGS DISPLAY.
C
          COMMON/PTHDAT/LDIS(1024),IFILL(1050),MAIND(50)
          COMMON/BACKG/IBDIS(256)
          COMMON/DATB/MGRID(375)
          EXTERNAL XCROSS
C
C
C                    INIT WORK DISPLAY
9Ø        LDIS(1)=Ø
          CALL DCHOOS(LDIS,1)
          CALL SETPT(Ø,Ø,Ø)
C
C                    INITIALIZE BACKGROUND DISPLAY
95        IBDIS(1)=Ø
          CALL DCHOOS(IBDIS,1)
          CALL SETPT(Ø,Ø,Ø)
C
C                    SET UP MAIN DISPLAY
C
          MAIND(1)=Ø
          CALL DCHOOS(MAIND,1)
          CALL DRAW(1,LDIS(1))
          CALL DRAW(1,IBDIS(1))
          CALL DRAW(1,XCROSS)
          MGRID(1)=Ø
          CALL GRID(MGRID(1),1)
          CALL BLANK(MGRID(1))
          CALL DRAW(1,MGRID(1))
C
C                    LINK TO PRTE MAIN DISPLAY USING
C                    FOG SAVE REGISTER 16. ONCE ONLY
C                    CODE.
          IF(IITIME.NE.Ø) GO TO 1ØØ
          IITIME=1
          CALL RCHOOS(16)
1Ø        CALL DRAW(1,MAIND(1))
          CALL DCHOOS(MAIND,1)
C
1ØØ       CONTINUE
          RETURN
          END

DOSPIP  V6A

>
```

Example 2

```
          SUBROUTINE SHOWIT
C
C         COMMAND:
C                   SHOW,NUM
C                   SHOW
C
C         DISPLAYS CEL OR PATH NUMBER <NUM> AND MAKES
C         IT THE CURRENT CEL. STYLUS HIT STEPS CEL NUMBER.
C          IF TYPED AND <NUM> OMITTED, SHOWS CURRENT CEL.
C
          DIMENSION CNAM(2),CDAT(2)
          COMMON/PTHDAT/IDDUM(2048),ICUR,NOCRV
          COMMON/ARGTP/IDDM(32),IDCOME
          DATA CNAM/5HSHOW ,5H        /
C
C                   GET CEL NUMBER
          NUM=ICUR
          IF(IDCOME .NE. 1) NUM=NUM+1
          CALL GETSI(2,NUM)
          IF(NUM .LT. 1 .OR. NUM .GT. 10) NUM=1
          ICUR=NUM
C
C                   DISPLAY CEL AND NUMBER
          ENCODE(10,CDAT,1) NUM
1         FORMAT(I3)
          CALL WDAT(CNAM(1),CDAT(1))
           CALL BLKDIS
          CALL SHOW(NUM)
C
          RETURN
          END

DOSPIP V6A

>
```

6. OVERLAYING A GRAPHICS APPLICATION PROGRAM

The PIGS run time environment is coded mostly in FORTRAN IV and would nearly fill the bottom 32K of PDP15 core (including device handlers and DOS) were it not overlayed. When overlayed, the run time environment uses about 10K of store (excluding device handlers and DOS). In order to achieve this core economy and allow application procedures to share PRTE and FORTRAN run time environment subroutines, graphic application programs must be loaded with PRTE routines using the overlay building program, XCHAIN.

6.1 Debugging the GAP

Overlaying a GAP with XCHAIN is a comparatively lengthy procedure requiring several minutes to complete the necessary library searches. For this reason, and because it is not possible to use DDT (Dynamic Debugging facility) with overlayed programs, it is wise to test application procedures separately from PRTE as much as possible. Otherwise, FORTRAN or MACRO-15 I/O statements must be used to track down errors, a lengthy process when compared with the debugging time required using DDT. Semantic errors in MENDEL programs are never difficult to find and do not require the GAP to be re-CHAINed unless the Jump Table was affected.

6.2 The Overlay Loader - XCHAIN

The DOS command XCHAIN brings into core the Atlas Laboratory version of the overlay loader, CHAIN. The latter is fully discussed in the PDP15 CHAIN manual. XCHAIN is described in PDP15 User Notes 2 and 3. Both references are essential to a proper understanding of the overlay process.

Briefly, XCHAIN outputs separate binary disc files containing overlay information, the core image of resident code, and the core image of each overlay (link). Collectively these files constitute an EXECUTE program which can be loaded and run using the DOS 'E' (EXECUTE) command.

In producing these files XCHAIN requires the following information:

(1) Name of execute program

(2) Library filenames and other load parameters

(3) Resident routine names

(4) Link names and the routines which reside in them

(5) A description of the manner in which links overlay each other

This information, the *overlay description*, may either be read from a disc file named CHAINX SRC, or input interactively using the system teletype. The discussion below assumes that a disc file description is used. Overlay information and a load map are produced on disc file CHAINX LST.

6.3 Overlaying PRTE

When overlaying GAP procedures with the run time environment, the PRTE links are kept separate from any GAP links required. Figure 6-1 illustrates how PRTE resident code and links would reside in core by themselves. There are five links which overlay each other in the PRTE link area. Each link has a separate function as described below:

Link name	Function
LK1	Presets constant common values
LK2	Opens, reads, and writes to the MNB file
LK3	Creates the PRTE display and initialises command source devices
LK4	Polls source devices for commands
LK5	Reconciles commands and parses text argument string

A complete load map for PRTE by itself is given in Appendix F and includes a list of the filenames of the routines included in each link and the resident code. The information required by XCHAIN to overlay PRTE is included prior to the load map as part of the CHAINX LST file.

The binary files which compose PRTE are all loaded from the PIGS library file, .LIBRP BIN. Appendix B includes an index of these routines and a brief description of their function. Note that in some cases the filenames do not match the entry point names of the subroutines. This does not influence overlaying, but XCHAIN load maps always give the *filenames* of routines loaded.

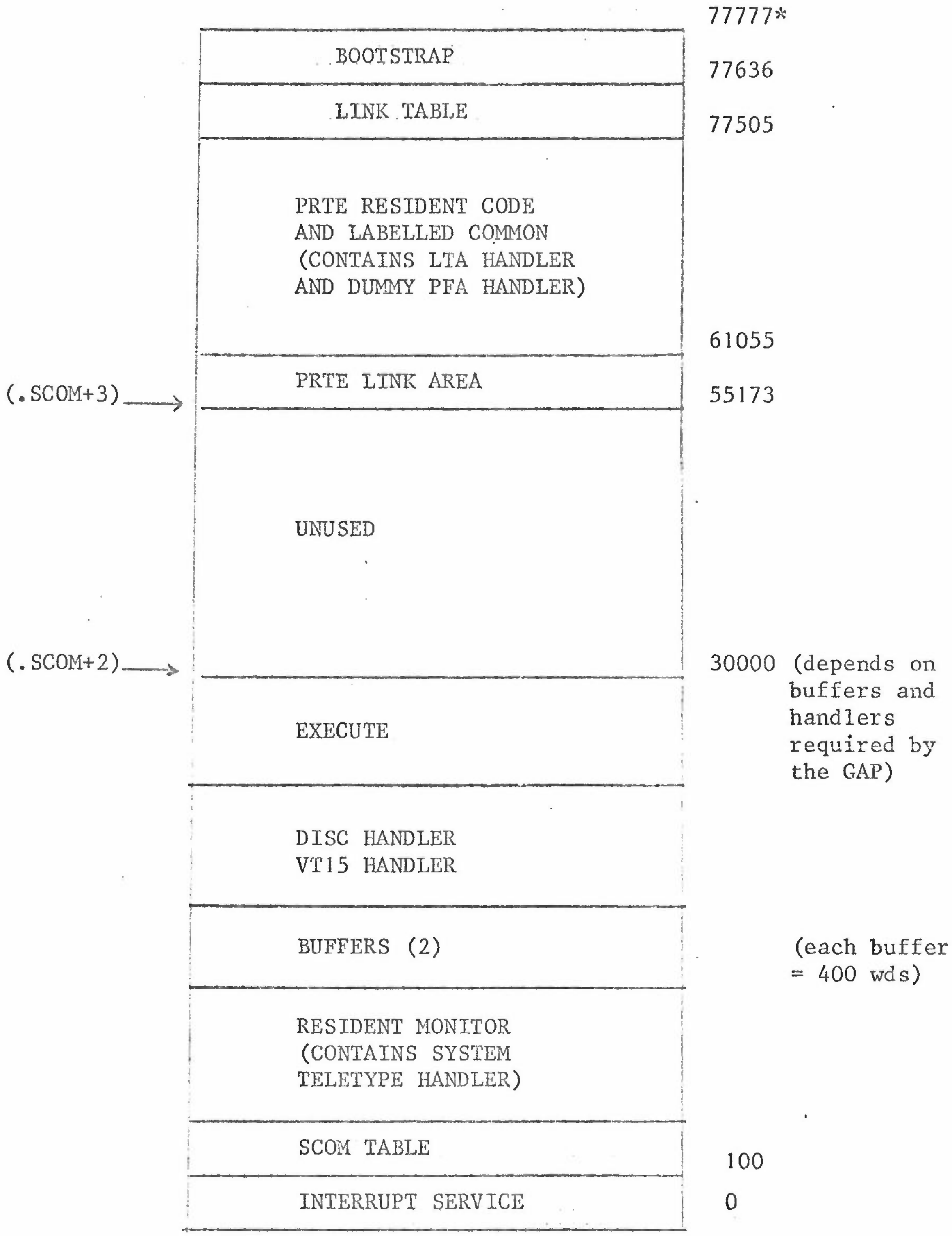
All PRTE overlay links are specified in the CHAINX overlay description using the library prefix, #, and the subroutine entry point name. The library prefix ensures that XCHAIN will load the named routines from file .LIBRP BIN as *external link components*, callable from other links or the resident code. Referencing an external link subroutine causes the link to be read into core. PRTE subroutines required within a link, but not specified in the description using the library prefix, are loaded from the library as *internal link components*. Internal link subroutines are *not* callable from other links or from the resident code.

The five links defined in the PRTE CHAINX description (Appendix F) overlay each other in a 1.5K area of core just below the resident code. Each link is loaded by a subroutine call to an external link component from the resident subroutine, PIGS. During the execution of a command, a maximum of 3 overlay changes in PRTE normally occurs:

LK4 to LK5	(polling to reconciliation)
LK5 to LK3	(if new menu required)
LK3 to LK4	(resume polling)

FIGURE 6-1 PRTE CORE ALLOCATION

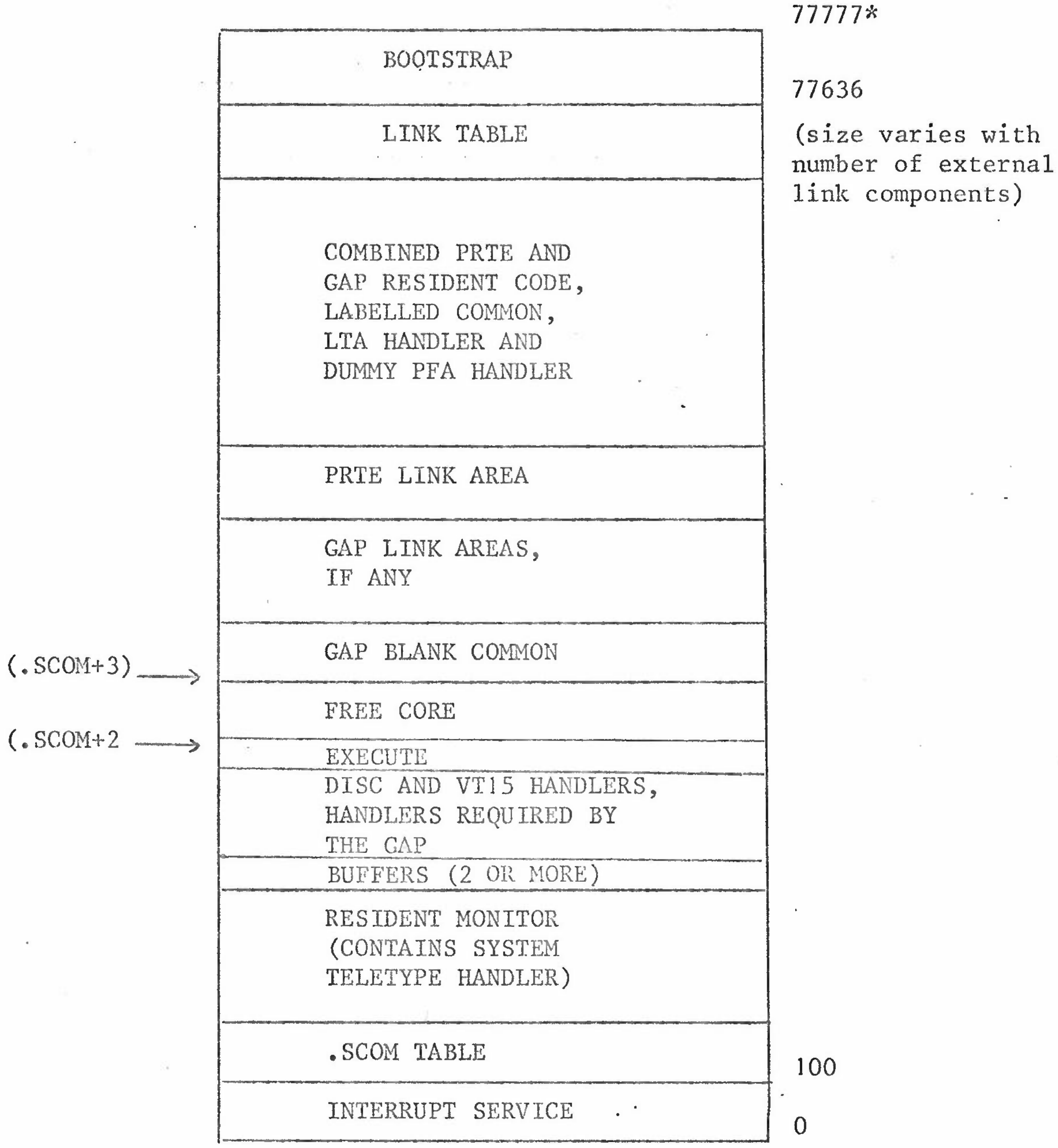
MEMORY MAP



* All constants above are octal radix
The upper 32K of core is unused under DOS

FIGURE 6-2 GAP CORE ALLOCATION

MEMORY MAP



77777*

BOOTSTRAP

77636

LINK TABLE

(size varies with number of external link components)

COMBINED PRTE AND GAP RESIDENT CODE, LABELLED COMMON, LTA HANDLER AND DUMMY PFA HANDLER

PRTE LINK AREA

GAP LINK AREAS, IF ANY

(.SCOM+3) →

GAP BLANK COMMON

FREE CORE

(.SCOM+2 →

EXECUTE

DISC AND VT15 HANDLERS, HANDLERS REQUIRED BY THE GAP

BUFFERS (2 OR MORE)

RESIDENT MONITOR (CONTAINS SYSTEM TELETYPE HANDLER)

.SCOM TABLE

100

INTERRUPT SERVICE

0

*All constants above are octal radix. The upper 32K is unused under DOS.

If no menu block I/O is required only 2 overlay changes occur. Since XCHAIN is roughly 5 times faster than the original CHAIN provided by DEC the delay is not noticeable.

6.4 Overlaying a Graphics Application Program

A GAP has approximately 10K of the lower 32K of core (exclusive of PRTE and the DOS operating system) in which to fit its own resident code and links. In the future it may be possible to use the upper 32K of store for free storage and display files, but not at present. More than the 10K of core is made effectively available by sharing some of the FORTRAN run time environment routines used by PRTE. Some graphics application programs, like PATH, will be able to run as purely resident code within the 10K unused by PRTE.

In practice overlaying a GAP with PRTE is a matter of slightly modifying the skeleton CHAINX description given in Appendix F to include the application's resident code and links. Appendix F includes the PATH CHAINX specification, while Figure 6-2 illustrates core allocation for PRTE and a GAP with overlays.

Normally GAP and PRTE links should not overlay each other. A CAP link may overlay the PRTE link area only if no link-resident PRTE routine is called before command execution is finished and control returns to PRTE. Otherwise the GAP link will overlay itself with disastrous results.

If there is no danger of a GAP link overlaying itself, application procedures may call certain of the PRTE resident or link subroutines.

PRTE routines CLOCK, CLOCHK, and DSCHED for example, are shared by PATH (refer to Appendix B for a list of shareable PRTE routines). In order to reference a link subroutine which is not normally an external link component, the desired entry point name must be added explicitly to its usual PRTE link definition and preceded by the library indicator, #, as has been done in the PATH overlay description (otherwise the subroutine will be loaded into the resident code area). Care should be taken to avoid duplicating reserved PRTE names with applications procedure or common block names. A list of these names appears in Appendix H.

6.5 Writing a GAP Overlay Description

Included below is a step-by-step description of how to build a CHAINX overlay description for a GAP. Responses to each XCHAIN prompting message are given. Only the responses should be inserted in the CHAINX file. Familiarity with both the CHAIN Manual and XCHAIN User Notices previously referenced is essential.

```
(1)  NAME XCT FILE
     >GAPNAM
```

The designer should reply with the name of the graphics application program.

```
(2)  LIST OPTIONS & PARAMETERS
       SZ,PAR,PAL,XSP,VTC/PIGDSP,DISER,PROMP,LAYDAT,COMTAB,OUTMOD/
```

It is useful, but not essential, to include the parameters SZ,PAR, and PAL. Parameter XSP must be included since this directs SCHAIN to search the PIGS library file, .LIBRP BIN, on the system disc area. The VTC option should also be included to ensure that no PRTE or GAP common block containing a display file crosses an 8K bank boundary. (Care must also be taken to see that application procedures containing display files or *indirectly displayed text strings* remain resident.) The designer should insert GAP common block names containing display files or displayed text between the slashes following the VTC option.

```
(3)  DEFINE RESIDENT CODE
     >GAPNAM,APLIC1,APLIC2,...
```

The designer must reply with the filename of the Jump Table (GAPNAM, always the same as the name of the GAP) and the filenames of any resident GAP procedures or block data programs (APPLIC1,APLIC2...). Combined files constructed using DOS utility programs PIP or UPDATE may be used instead of naming each application procedure separately. File CPATH, mentioned in the PATH overlay description, is such a combined file. Any subroutines required by the GAP resident code will be loaded either from the user library file (.LIBR5), the PIGS library (.LIBRP) or the system library (.LIBR). If not declared as external link components, such routines will also be resident.

```
(4)  DESCRIBE LINKS & STRUCTURE
     >LK1=#COMVAL
     >LK2=#YYSTRT, OPMNB, RDMNU, WTMNU
     >LK3=#DISINT
     >LK4=#POLL
     >LK5=#RESOLVE
     >     DEFINE GAP LINKS, IF ANY
     >LK1:LK2:LK3:LK4:LK5
     >     DEFINE GAP OVERLAY STRUCTURE, IF ANY
     ><space><altmode>
```

These replies describe both the PRTE and GAP overlay structure. The PRTE links should only be altered to make a normally internal link component into an external link component. Any names desired except for LK1,LK2, LK3,LK4, and LK5 may be used as GAP link names.

7. DOCUMENTING A GRAPHICS APPLICATION PROGRAM

One of the happy consequences of following any programming convention, such as the argument-passing scheme used by PRTE, is that libraries of useful programs may be compiled, saving programming time and effort. One of the unhappy consequences is that someone must describe in everyday language what is in such libraries. Graphics applications programs which have been debugged and are ready for general use should be documented according to the guidelines presented in this chapter.

7.1 PDP15 Libraries at ACL

There are 5 libraries which may be of use to the GAP designer at ACL:

(1) The System Library

Largely FORTRAN-oriented, this library exists on the system disc area as .LIBR BIN. Its contents are documented in the FORTRAN IV Operating Environment Manual. No sources are available.

(2) The PDP15 Routine Library

Consisting of useful FORTRAN and MACRO routines, the library is documented in a manual of the same name. Source files for the routines exist on DECtapes 1000-1099. The PDP15 Routine Library also contains useful GAP procedures and shareable PRTE routines.

(3) The PIGS Library (see Appendix B)

This library exists as file .LIBRP BIN on the system disc area and contains PRTE subroutines and a few commonly-used GAP procedures. A source listing of the library is given in the GAP Library. Source files exist on DECtapes 156-159.

(4) The DECUS Library

Consisting largely of systems programs, the DECUS Library also contains some very useful FORTRAN-callable subroutines obtained from the DEC user's organisation. Source files and binaries exist on DECtapes 50-99.

(5) The GAP Library

All graphic applications implemented using PIGS are contained in this library, on DECtapes 150-199. The documentation consists of program listings and a manual containing descriptions of each GAP.

Documentation for all of the above libraries is kept in the bookshelves by the PDP15.

7.2 GAP Documentation

Documentation of a debugged graphics application using PIGS includes a set of program listings and a written description of the use and internal operation of the package. The documentation should be submitted, along with DECtape source files, to the PDP15 operator on duty at the 1906A console for typing and distribution.

7.3 GAP - Written Description

Each of the numbered topics described below should be included in the GAP written description. The description should be written on lined paper, in ink. The description of PATH in Appendix A should serve as an example.

(1) Name

(a) Name of Application
(b) GAP designer's name
(c) Date
(d) Filenames and location (DECtape number) of EXECUTE files, MENDEL source file, CHAINX LST file, GAP source and binary files.

(2) Purpose

A *brief* description of the application problem and solution, allowing rapid scanning through the library.

(3) Loading Procedure

(a) Physical device readying (such as BSI start-up, DECtape mounting, DMAC boards required)
(b) Usual device assignments required
(c) Buffer assignments needed
(d) A teletype listing example of 3(b) and (c)

(4) Description

(a) Problem description
This section should elaborate on the particular problem which the GAP approaches, the methods used in the solution, and general information on how to use the package
(b) Input and Output
Should include a description of device input and output formats and their meaning, DAT slots used
(c) Internal Data Base
Should describe the structure and meaning of problem area data base arrays or common blocks. The relationship to the GAP display should be included, if applicable
(d) Display Structure
Should include a description of the GAP display file structure and the meaning of any special displays used. Sample sketches of the CRT may be helpful

(5) MENDEL Source Listing

Teletype print-out included here to clarify section (6), below. The source code should be well commented.

(6) Menu and Command Description

This section follows the general outline of the MENDEL source listing of section (5) and discusses the function of the commands which comprise the GAP. The following subsections should be included:

(a) Global commands (displayed)
(b) Global commands (non-displayed)
(c) Menus
    1. Menu name
    2. Exit command
    3. Enter command
    4. Pushbutton commands
    5. Local commands

Each command description should deal with the command's function, data area, and the following data about its arguments:

Function
Type and permissible values
Default value if omitted

(7) Error Messages

Should include an explanation of each error number and the appropriate action to be taken by the operator.

(8) Example

Should include the commands necessary to perform one full interactive problem solution or loop converging on a solution.

7.4 GAP - Listings

When submitting an application for inclusion in the GAP library, the following lineprinter listings should be included:

(1) MENDEL source
(2) CHAINX LST for the GAP
(3) GAP source for all procedures

8. FUTURE ENHANCEMENTS

During the design of PIGS a number of ideas for useful software have arisen which are not yet implemented, either because of lack of time or because they were thought better left until more experience with the first system was gained. This chapter merely catalogues some of the ideas for future evaluation.

8.1 MENDEL Editor for PRTE

The MENDEL editor has been written in such a way that its inclusion as an overlay of PRTE would be simple. Inclusion of the editor in PRTE would allow reorganisation and addition of menus and commands at run time to meet unexpected problem requirements. The utility of such a system depends on being able to write applications procedures using either FOCAL or command macros (see 8.6).

8.2 Protection against System Crashes

Things fall apart! The wise GAP designer will provide a data base dumping command as part of his package. PRTE could dump automatically at regular intervals using its scheduling mechanism if it knew the name of the GAP dumping command. Variables in the MNB file header have already been allocated for dump command name and time interval, but are not used in PIGS V2.

A useful feature for debugging, backup, and evaluation is a session log of executed commands and errors. Entries in the log would include a type code, time, and the ASCII command string or message. The log would probably exist as a disc file, portions of which could be listed from PRTE.

8.3 New Command Sources

Commands are already available within MENDEL to reference the DMAC pen follower and the ICL 1906A computer as command sources. When suitable handlers for the two devices are completed, incorporating them into the PRTE polling loop should be trivial.

8.4 Core Management

The GAP designer currently has only about 1ØK of the lower 32K of store in which to squeeze procedures, data base, and display files while the upper 32K sits empty because the DOS loader cannot access it. It is possible, however, to place data and display files in upper store and reference them from lower core. FOG could be slightly modified to assemble display files in the upper 32K of store and start them running. Some sort of fixed block size core management routines would be required.

A second possibility for utilising upper store is that the overlay builder and loader could be slightly modified to relocate execute programs into upper(or lower)core. PRTE would communicate with a GAP in lower store via special subroutine calls. The Jump Table would reside in lower store with the GAP. Such a system would allow the use of DDT with GAP application procedures running under PRTE. Run time PRTE and FORTRAN procedures could not, however, be shared. The loading time necessary for a GAP would be drastically reduced.

8.5 PIGS under other Operating Systems

PRTE was designed specifically for the DOS15 operating system. Although FORTRAN coded, its operational philosophy is heavily dependent on the use of a disc-based, single-user executive.

Because it polls for commands rather than waits for interrupts, conversion of PIGS for a multiprogramming system (such as the Resource Sharing Executive, RSX) would require design changes to allow efficient CPU usage. PIGS could be converted for interrupt-driven command input, or, less efficiently, time slicing could be incorporated in the polling loop to give other system users better response.

Although PIGS was designed to consist of shareable subroutines, it is not re-entrant and therefore would require a major overhaul to service multiple users. For the same reason, splitting PRTE and the GAP into two processes is a non-trivial task (but very similar to using PRTE in upper core under DOS).

8.6 Source Languages for Application Procedures

Within an interaction session an operator may find he spends much of his time repeating a particular set of commands with only slightly varying parameters. In some cases the GAP designer may have foreseen such a situation and provided a single command to do that job. In many cases however, the command loop in use is a function of the particular problem, and cannot therefore be dealt with specially in the general problem approach taken by the GAP. It seems that a language for defining simple application procedures at run time would lessen operator time spent in such problem loops. An interpretive language is the obvious candidate, obviating the need for compilation and loading.

PDP15 FOCAL (like BASIC) is one language under consideration. The most sensible candidate, however, is a command macro facility within PRTE itself. Macros would be stored as text files on the disc in PRTE command format. Arguments could be bound before execution of the macro using a set of special argument buffers, or during execution by using the "left" argument facility (see 8.8). Definition of macros would be possible using the PDP15 text editor, or by storing GAP commands when a PRTE flag is set. Macros could be used to define new application commands if the MENDEL editor were included in PRTE.

8.7 Messages

A well-written graphics application program may contain many error and prompting messages. Since these messages require much core space and rarely change, they might easily be kept as random access disc files. Individual messages could then be referenced by number.

When learning to use a GAP, it would be useful at first to have more guidance than the occasional prompting area message. A help file could include useful information about each command's function and arguments. Help information could be displayed by the operator as desired.

8.8 Argument Input

Included in MENDEL, but currently undefined, is the ARGET statement. If implemented, the ARGET command would open a special MNB menu block for definition. This menu would contain a number of standard commands for defining arguments. Each command could be used either to fill a special argument buffer (see 8.6) or to replace a *left argument*. Left arguments could be indicated in a command by using the character, * , and would cause the ARGET menu to be temporarily activated. Typical ARGET commands might activate procedures to retrieve the X or Y coordinates of a point specified with the stylus, the distance between two points, some text, or the time. All such commands would return text strings to either fill an argument buffer or replace the character,*, in the command string.

A second useful argument input facility would be a non-parse option for GAP commands. If the number-of-arguments-parameter to the MENDEL COM command were specified as -2 for a particular GAP command (-1 means indefinite number of arguments), PRTE would never parse that command's argument string but simply pass the entire command string to the associated application procedure. Only left arguments in the argument string would be detected and evaluated. This facility would be useful for transmitting commands with non-PRTE syntax to programs in the linked ICL 1906A computer.

APPENDICES

APPENDIX A - EXAMPLE DOCUMENTATION, PATH

(1) Name

PATH

by W D SHAW 9/16/74

Filename	Function	DECtape location
PATH XCT		155
PATH XCU		155
PATH L01		155
PATH L02	PATH EXECUTE	155
PATH L03	FILE	155
PATH L04		155
PATH L05		155
PATH MDL	MENDEL SOURCE	155
CHAINX LST	LOAD MAP	155
PINIT SRC		163
STYDT2 SRC		163
DATACL SRC		163
BLKDIS SRC		163
INITD2 SRC		163
PAGECL SRC		163
DRAWC SRC		163
REDRAW SRC		163
DATB SRC	GAP SOURCE FILES	163
CEL SRC		163
DRAWB SRC		163
BACKGR SRC		163
PLAYBA SRC		163
SHOW SRC		163
FRATE SRC		163
GRIDAT SRC		163
SHOWIT SRC		163
LTPEN1 SRC		158
CPATH BIN	UPDATE FILE	155

(2) Purpose

Definition and playback of simple animated sequences.

(3) Loading Procedure

Before starting PATH, the VWØ1 sparkpen should be turned ON and the LK35 keyboard OFF. No special DAT slot assignments or extra I/O buffers are required.

A

```
DOS-15 V2A
$LOG MOUNT DECTAPE 155 ON UNIT 1
$KEEP OFF

$LOGIN SCR

$PIP

DOSPIP V6A

>C DK←DT1


>↑C

DOS-15 V2A
$VT OFF


 E PATH


PIGS V2
>PATH
```

(4) Description

A serious problem in both conventional and computer animation is finding a natural means of describing character movement. One convenient solution is to mimic a motion by using the sparkpen to draw it with the speed and positional changes desired, while the computer digitizes points on the motion curve (*path*) at some constant frequency. If the character (*cel*) is then moved by the computer from point to point at the same frequency a simple animated sequence is generated which closely resembles what the operator wanted with no need to define internal computer coordinate systems or key frames. Unfortunately, only very simple animation can be described in this manner: no distortion, scaling, or rotation is possible.

PATH uses the above approach to allow the operator to animate simple hand-drawn cels against a fixed background. Although the lightpen may be used to select commands, only the sparkpen should be used for drawing since it is time independent. Initially the sparkpen and LK35 keyboard are the active stylus and keyboard devices, respectively.

Within the computer, cels and paths are stored identically and are, in fact, interchangeable. Each drawing is given a unique number between 1 and 10 when it is defined. Cels and paths are always referenced in PATH commands by their identifying number. The definition of each drawing is kept in common block PTHDAT and may be displayed when desired. The background definition, by contrast, exists only as a display file in common block BACKG. Both of these important data areas are described below:

A

```
COMMON/BACKG/IBDIS(256)
```

IBDIS(256)   Background display file.

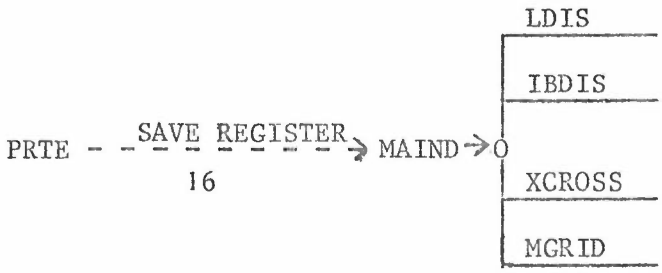
```
COMMON/PTHDAT/LDIS(1024),IPTHX(512),IPTHY(512),
ICUR,NOCRV,ICB(10),ICT(10),IBOT,ITOP,LOCSET,ILST
```

LDIS(1024)	Displays current cel or path.
IPTHX(512),IPTHY(512)	Coordinates of all drawings defined. (IPTHX(*N*),IPTHY(*N*)) is either a point in a connected curve or the starting point of a connected curve. In the latter case the X coordinate will be negated.
ICUR	Current cel number for playback.
NOCRV	Current path number for playback.
ICB(10)	ICB(*N*) points to the first coordinate pair in (IPTHX,IPTHY) of the definition of drawing *N*.
ICT(10)	ICT(*N*) points to the last coordinate pair in (IPTHX,IPTHY) of the definition of drawing *N*.
IBOT	Next free location in (IPTHX,IPTHY).
ITOP	Last free location in (IPTHX,IPTHY).
LOCSET	Unused.
ILST	Number of last drawing defined.
MAIND(50)	Main display file.

```
COMMON/FRATE/IFPS
```

IFPS	Playback rate in frames per second.

The PATH display file structure is relatively simple. Array MAIND is the main display file containing DRAW's to subfiles LDIS,IBDIS,XCROSS, and MGRID. Display file IBDIS is used only by the DRAWB command while LDIS is used by the DRAW command and by subroutine SHOW to display a cel or path. File XCROSS defines the lightpen tracking cross while file MGRID defines a rectangular grid. The last two files are used by subroutine STYLI and are initially blanked.



A

(5) MENDEL Source Listing

```
/********** PATH MENU DEFINITION **********
/
/
/ MENDEL PROGRAMS TO DESCRIBE THE COMMAND STRUCTURE OF PATH,
/ A PROGRAM FOR DOING SIMPLE INTERACTIVE COMPUTER ANIMATION.
/ PATH ALLOWS AN OPERATOR TO INPUT A CURVE (CALLED A PATH OR
/ P-CURVE) AND A CHARACTER (CALLED A CEL) USING THE TABLET
/ OR LIGHTPEN.  HE MAY THEN CAUSE THE CEL TO MOVE ALONG THE
/ PATH WITH THE SAME VELOCITY CHANGES WITH WHICH IT WAS
/ DRAWN.
/
/
/********** DEFINE  PATH  INITIALIZATION  **********
/
/ MENUS WILL BE IN BINARY FILE   PATH MNB.
/ RELOCATEABLE BINARY JUMP TABLE WILL BE IN FILE  PATH BIN.
/
CREAT,PATH,5,62,24                   /ROOM FOR 5 MENUS, 62 SUBBRS, 24 GLOBAL.
BIGBT                      /USES LARGE LIGHTBUTTON DISPLAY
ABREV,TRUE                 /USE 5 CHARACTER COMMAND ABBREVIATIONS.
KEYB,LTA                             /LK35 KEYBOARD INITIALLY SELECTED
STYLS,VWA                            /VW01 SPARKPEN INITIALLY SELECTED.
DELAY,200                            /BLINK LIGHTBUTTON SELECTED FOR 200 MILLISEC
SAVE,FALSE                 /DO NOT WRITE OUT  SVMNU'ED  MENUS.
/
/
/********** DECLARE MENU NAMES **********
/
MNDEC,PATH,BACKG                     /PATH AND BACKG ARE ONLY MENUS BUT THE CREAT
                           /COMMAND ALLOWED ROOM FOR 5 .
/
/
/********* DECLARE SUBROUTINE NAMES **********
/
SBDEC,USESTY,DATACL,DRAWC,SKEDL,DSKED                  /DECLARE EACH SUBROUTINE
SBDEC,PLAYBA,QUIT,REDRAW,PINIT                         /NAME ASSOCIATED WITH
SBDEC,USEGRI,USEKEY,BACKGR,SHOWIT                               /SOME COMMAND.

/
/
/********** DEFINE GLOBAL COMMANDS **********
/
/ THE FIRST SIX DEFINED APPEAR AS LIGHTBUTTONS.
/
GLOBL                      /START GLOBL DEFINITION
COM,USESTYLUS,1,DO,USESTY,,,VWA              /TO SELECT OTHER STYLUS.
COM,USEKEYBRD,2,DO,USEKEY,,,LTA              /TO SELECT OTHER KEYBRD.
COM,SHOW,1,DO,SHOWIT                         /DISPLAY CEL OR PATH.
COM
COM,QUIT,,DO,QUIT,,,'!!!!!!!!!'              /EXITS PROGRAM PATH.
COM
COM,SKEDL,5,DONOW,SKEDL                      /SCHEDULES A COMMAND
COM,DSKED,1,DONOW,DSKED             /DESCHEDULES A COMMAND
/
/
/********** ARGET **********
/
/ NOT CURRENTLY IMPLEMENTED, BUT COMMAND BELOW MUST BE PRESENT.
/
ARGET
/
/
/
```

```
/
/********** GROUP COMMANDS INTO MENUS **********
/
/ EACH COMMAND MAY BE LINKED TO A DECLARED SUBROUTINE.
/
MENU,PATH                       /START MENU NAMED PATH.
ENTER                   /DEFINES MENU NAME AND ENTRY ROUTINE
COM,PATH,,DO,PINIT,,,' INIT' /WILL APPEAR BELOW 'MENU' IN
                        /CONTROL AREA.  ALSO CLEARS DATA BASE.
/   NO EXIT COMMAND FOR TOP LEVEL MENU
/
/
PUSHB,1                 /DEFINE PUSHBUTTON 1.
COM,PENACTIVE,,DONT     /THESE 3 ARE ONLY A DISPLAY USED BY A
COM,SETPOINT,,DONT              /COMMAND SUBROUTINE. NOT ASSOCIATED
COM,ACCEPT,,DONT                /WITH A SUBROUTINE.
/
LOCAL                   /DEFINES LOCAL COMMANDS.
COM,DRAW,2,DO,DRAWC,,,' 1'      /DRAW A CEL OR PATH.
COM,PLAYBACK,3,DO,PLAYBA,,,' 1,1' /PLAYBACK THE MOTION AT A
                                /GIVEN RATE.
COM,REDRAW,1,DO,REDRAW,,,' 1'   /REDRAW A CEL OR PATH.
COM,DATCLEAR,,DO,DATACL         /SAME AS COMMAND PATH.
COM,USEGRID,,DO,USEGRI,,,' NO'          /USE GRID WHEN DRAWING.
COM                                     /NULL COMMANDS.
COM
COM,BACKG,,,,GO,BACKG                   /ACTIVATE NEW MENU, BACKG.
/
/
/
MENU,BACKG
ENTER
COM,BACKG                               /MENU NAMED BACKG. NO ENTRY COMMAND
/
PUSHB,1
COM,PENACTIVE,,DONT
COM,SETPOINT,,DONT
COM,ACCEPT,,DONT
/
EXIT
COM,PATH,,,,GO,PATH             /RETURN TO MENU PATH.
/
LOCAL
COM,DRAWB,,DO,BACKGR            /COMMAND TO DEFINE BACKGRND.
/
/
/********** TERMINATE MENDEL PROGRAM **********
/
END,PATH                        /STARTING MENU IS  NAMED PATH.
```

A

(6) MENU and Command Description

(a) Global Commands (displayed),PATH

USESTYLUS,IDEV

Uses procedure USESTY (from .LIBRP) to change the active stylus device. The tablet must be ON otherwise an IOPS4 message will occur. The data area names the active device, 'VWA' or 'LPN'.

IDEV	Ø	Sparkpen active
	-1	Lightpen active
	omitted	Sparkpen active

USEKEY,KDEV,LECHO

Uses procedure USEKEY (from .LIBRP) to change the active keyboard device. The data area names the active device, 'TTA' or 'LTA'

KDEV	Ø	TTA keyboard active
	1	LTA keyboard active
	omitted	No change in device
LECHO	Ø	No echo on TTA
	-1	Echo on TTA
	omitted	No change in echo

SHOW,NCEL

Causes a cel or path to be displayed and makes it the current cel number. The data area shows the number of the displayed drawing. If selected using the active stylus, the current cel number is incremented and displayed.

NCEL	Ø-1Ø	Number of the cel to be made current and displayed. If NCEL > 1Ø,NCEL=1 is assumed.
	omitted	The current cel number is displayed.

QUIT

Causes exit to DOS operating system.

(b) Global Commands (non-displayed), PATH

SKEDL,CNAME,ITER,INTM,INTS,INTP

Uses procedure SKEDL (.LIBRP) to schedule a command for repeated execution at a given time interval.

CNAME		Command name. May not be omitted.
ITER	≥Ø	Number of times command will be selected.
	<Ø	Command to be repeated indefinitely or until descheduled by the operator.
	omitted	ITER = - 1
INTM	≥Ø	Repeat interval in minutes
	omitted	Ø assumed
INTS	≥Ø	Repeat interval in seconds
	omitted	Ø assumed
INTP	≥Ø	Repeat interval in clock pulses (20 millisec)
	omitted	Ø assumed

The repeat interval is INTM+INTS+INTP.

DSKED,CNAME

Uses subroutine DSKED (.LIBRP) to deschedule a command.

CNAME	string	Name of command to be descheduled.
	omitted	Deschedule all commands in the clock scheduling buffer.

(c) MENU Commands

MENU,PATH

ENTER

PATH

The associated entry procedure, PINIT, initialises the GAP display and data base. Unlike DATCLEAR, it also clears the background display.

EXIT

There is no exit command for PATH.

PUSHB

Only pushbuttons 1-3 are used by PATH and all are inactive until procedure STYLI is entered via the DRAW or DRAWB command. STYLI reads the pushbuttons directly, using them as described below.

PENACTIVE

This lightbutton appears when STYLI has been entered via the DRAW or DRAWB command. It turns the active stylus ON or OFF. The data area reads 'YES' or 'NO'. Pushbutton 1 must be pressed before starting, and after terminating a drawing.

SETPOINT

This lightbutton appears when STYLI has been entered via the DRAW or DRAWB command and the lightpen is the active stylus device. The data area reads 'YES' or 'NO' if the next coordinate pair read will start a new connected curve, or not. Thus pushbutton 2 can be used to move the tracking cross around without 'drawing'.

ACCEPT

This lightbutton appears when STYLI has been entered via the DRAW or DRAWB command and the grid option is in use (see USEGRID command, below). Pressing pushbutton 3 causes the data area to flip between XXXXXXXXX and blank. The current stylus position is mapped onto the closest grid point and accepted as the next coordinate of the drawing.

LOCAL

DRAW,IBUBS,NUMBER

If selected with a stylus device, this command increments the current path number by 1 (it is initially Ø) and enters procedure STYLI to accept a drawing. IBUBS, the minimum distance between consecutive accepted points, is usually Ø. (It *must* be zero for a good path definition.) The data area shows the drawing number being defined.

A

IBUBS	number	Bubble size, as above A value of 1Ø is useful for thinning a cel drawing, but Ø must be used for a path.
	omitted	Unchanged
NUMBER	1-1Ø	Number of drawing to be defined
	omitted	Current path number used

PLAYBACK,ICUR,NOCRV,IFPS

Causes the current cel to be animated using the current path. The *first* point of the cel matches the points in the path.

ICUR	1-10	Drawing number to be made current cel number
	omitted	Current cel number used and unchanged
NOCRV	1-10	Drawing number to be made current path number
	omitted	Current path number used and unchanged
IFPS	1,2,5,10,25,50	Playback rate, frames per second
	omitted	Use current playback rate (initially 25 frames per second)

REDRAW,NOCRV

Same as DRAW command, but does not increment the current path number. This command should be used to redefine the last drawing.

NOCRV	1-10	Number of drawing to be redefined
	omitted	Redefine the last drawing

DATCLEAR

Clears the data base and drawing display.

USEGRID

Uses procedure USEGRI (.LIBRP) to select the grid option of STYLI. All coordinates entered into the current drawing are first mapped onto the nearest point of a displayed rectangular grid if the data area of USEGRI reads 'YES'. Selecting the command again resets the option to 'NO'.

BACKG

Activates menu BACKG.

MENU,BACKG

<u>ENTER</u>

BACKG

<u>EXIT</u>. No procedure associated, merely names the menu

PATH

GO's to menu PATH, no exit procedure associated

<u>PUSHB</u>

There are three pushbutton commands, identical to those in menu PATH

LOCAL

DRAWB
Uses procedure STYLI to define the background display. There are no arguments to this command. The drawing bubble size is temporarily set to 1Ø, then reset to its old value.

(7) Error Messages

Number	Description
203	Drawing data base overflow The data base must be cleared using PATH and the drawing repeated
204	Drawing display file overflow Again, command PATH should be used

(8) Example

The following commands form a typical interactive loop in defining an animated sequence using PATH. (Typed commands are preceded by >. Comments are preceded by / and should not be typed.)

```
    DRAW                              /Define cel 1
    PENACTIVE                         /Press pushbutton 1 to start
/ At this point cel 1 is drawn with the sparkpen
    PENACTIVE                         /Press pushbutton 1 to quit
    DRAW                              /Define path 2
    PENACTIVE                         /Press pushbutton  1 to start
/ Draw path as you want the cel to move
    PENACTIVE                         /Quit drawing
    PLAYBACK
/ At this point the current cel, 1, will move along the current path, 2,
  at 25 frames per second with the motion desired.
    REDRAW                            /Path 2 was not as
                                      /desired, this will
                                      /redefine it
    PENACTIVE                         /Start drawing
/ Now redraw path 2
    PENACTIVE                         /Quit drawing
/ A background may be defined by changing menus to BACKG
    BACKG                             /Change menus
    DRAWB                             /Define the background
    PENACTIVE                         /Start drawing
/ Now draw a background.  The stylus
/ must be moved at least 1Ø rasters before
/ any lines will appear.
    PENACTIVE                         /Quit drawing
    PATH                              /Return to menu PATH
    PLAYBACK                          /And playback cel 1, path 2
/ Now, to type, turn the sparkpen OFF and the
/ LTA keyboard ON to avoid interference
/ Playback the animated sequence in slow motion:
  >PLAYBACK,1,2,1Ø                    /1Ø frames per second
  >QUIT                               /Exit to DOS
```

APPENDIX B - PIGS LIBRARY

The PIGS library is a binary file, .LIBRP BIN, residing on the system disc area. It contains all the relocateable binaries necessary to overlay both the PIGS run time environment and the MENDEL editor-assembler. It also contains special PRTE control and application routines useful to the GAP designer.

All of the subroutines in the library are FORTRAN-callable, but some may be used only by PRTE. The routines are listed in 4 categories:

(1) Non-shareable PIGS routines
(2) Shareable PIGS routines
(3) Special PRTE routines
(4) Application procedures

Category (3) routines are fully described in Tables 5-1, 5-2, and 5-3 of this manual. Category (2) and (4) subroutines are documented in the PDP15 FORTRAN library. A list of all PIGS library routines are included here, but only Category (1) routines are discussed.

B

PIGS LIBRARY INDEX

(1) Non-Shareable PRTE

PFDUM
ADDCM2
BLANKS
BUTHT2
BUTINT
CMFIND
CMINT
COMVAL
DISINT
DOTINT
DOTMV2
ERRMS4
JPSTAR
JUMPT2
LAYOU2
LTA
NMNU2
NXTCHR
OPMNB2
PARG
PIGDNT
PIGFIL
PIGS2
PINT
PNUMB
PRCOMT
PRIN1
PROMPT
PRIN1
PROMPT
PSTRNG
RDMNU2
RESLIB
SYSDN2
TERMIN
VTERR2
WINK
WMNU
WTMNU2
YYCLIN
YYDCD
YYFLA
YYSTRT

(2) Shareable PRTE

BUTDIS
CBUTHT
CLOCHK
CLOCK
CVICH
DSCHED
FNAME
GRID
IBITS
INBITS
KEYIN3
KEYS
KEYST3
LPHIT
LTA
LTPEN
LTPEN1
MESDIS
MESINT
MSTR
MVC
PBHIT
POLL3
RESLV2
SCHED
STYLI2
TABLET
TARGS2
TIMEP
WRTUM
YYPRLN

(3) Special PRTE

ACTCM
CLERR2
CLPRM
DACTCM
DSKEDU
DISERR
ERP
GETCH2
GETDI2
GETDP2
GETFIL
GETLOG
GETSI2
GETSR2
PROMPT
PUTCH2
PUTDI2
PUTDP2
PUTSI2
PUTSR2
QUIT
WDAT

(4) Application Procedures

DSKED
SKEDL
USECRI
USEKEY
USESTY

(1) NON-SHAREABLE PRTE ROUTINES

SUBROUTINE NAME	FILENAME (IF DIFFERENT)	OVERLAY	DESCRIPTION
PFDUM		RESIDENT	Dummy DMAC handler required by KEYS
ADDCOM	ADDCM2		Adds a command node to the menu being defined
BLANKS		LK5	Removes blanks from command string input (TARGS)
BUTHIT	BUTHT2	LK4	Checks a simple lightbutton file for hits
BUTINT		LK3	Creates the PRTE lightbutton files using FOG
CMFIND			Searches the current Command Table for a particular command name
COMINT		LK2	Opens MNB file and uses header to initialise PRTE Command Table
COMVAL		LK1	Presets PRTE constants in common blocks
DISINT		LK3	Creates the PRTE display using FOG and initialises active command source devices
DOTINT		LK3	Creates tracking dot display
DOTMV2		LK4	Repositions the tracking dot on the display
ERRMES		RESIDENT	PRTE error message handling, contains entry point ABORT
LTA.		RESIDENT	Handler for LK35 and TEKTRONIX keyboards
LAYOUT		LK3	Creates frame and MENU, EXIT titles for PRTE display
NMNU	NMNU2	RESIDENT	Activates a new MENU, given an MNB file menu block number
NXTCHR		LK5	Retrieves the next character from the command string input (TARGS)
OPMNB	OPMNB2	LK2	Opens MNB file for reading or writing 250-word blocks
PARG		LK5	Parses the next argument in the command string
PIGDNT		LK3	Creates the main PRTE display file and sets FOG save register 16
PIGFIL		LK2	Reads MNB filename from TTA
PIGS	PIGS2	RESIDENT	Contains main PRTE control loop. Calls all overlays into core
PINT		LK5	Parses a signed integer argument in the command string
PNUMB		LK5	Parses a number argument in the command string
PRCOMT			Decodes a menu block and writes it to the Dump file
PRIN1		RESIDENT	Writes a single character to the teletype
PROMPI		LK3	Creates the prompting area display file using FOG

Cont'd

SUBROUTINE NAME	FILENAME (IF DIFFERENT)	OVERLAY OR MENDEL?	DESCRIPTION
PSTRNG		LK5	Parses a string argument in the command string
RDMNU	RDMNU2	LK2	Reads a menu block from the MNB file into the Command Table
RESLIB		RESIDENT	Causes certain system library routines to be resident
SYSDNT	SYSDN2	LK3	Creates the PRTE display and initialises the source devices (called by DISINT)
TERMIN		LK5	Searches the command string for a terminating character
VTERR	VTERR2	RESIDENT	FOG error handling routine for PRTE
WINK		RESIDENT	Starts a lightbutton with a given name register value blinking on and off
WMNU			Used by WDAT to write a message in a command data area
WTMNU	WTMNU2	LK2	Writes a menu block to the MNB file
YYCLIN			Decodes a command node into a 5/7 ASCII representation
YYDCD			Unpacks a command node
YYFLA		RESIDENT	Blinks a lightbutton for a given time interval
YYSTRT		LK2	Parses the MNB filename

APPENDIX C - PIGS COMMON BLOCKS

A list of PRTE, MENDEL, and PIGS library labelled common blocks is given below, followed by a description of their contents and function.

MENDEL COMMON BLOCKS	PRTE COMMON BLOCKS	Other .LIBRP COMMON BLOCKS
ARGTP	ARGTP	GRIDAT
CARG	CARG	PSHBUT
CHARS	CHARS	STYDAT
CODBIN	COMTAB	
COMTAB	DARG	
DARG	DISER	
ERRCON	ERRCON	
MDLBUF	HITBUT	
MDLUAR	LAYDAT	
PAGMNU	MNUDAT	
	OUTMOD	
	PAGMNU	
	PIGDSP	
	PROMP	
	PUSHB	
	SCHARR	
	STYLS	
	TIMEB	

APPENDIX C - PRTE COMMON BLOCKS

BLOCK NAME	VARIABLES	DESCRIPTION
ARGTP		Argument source and type
	NCOM	Index in COMTAB of selected command
	NOARGS	Index of last specified argument
	ITYPE(15,2)	ITYPE (N,1) is the type of the argument *N*
		∅ Argument *N* omitted
		1-255 String argument, number of characters
		256 Number argument
		512 Left argument (not implemented, treated as omitted)
		ITYPE (N,2) For number or string arguments, a pointer to the argument value in common block DARG or CARG. For number arguments ITYPE (N,2) is an index in the RARG array. For string arguments it is a character index in the SARG array. (See DARG and CARG, below.)
	IDCOME	Source device code for the selected command
		∅ Command called from a GAP procedure using argument-putting routines
		1 Keyboard
		2 Lightbutton
		4 Pushbutton
		8 Real Time Clock
CARG		Character argument buffer (see ARGTP)
	SARG (15)	Contains character strings input as arguments
	ISP	Next free character position, indexed from ∅, in SARG
CHARS		5/7 ASCII character codes used by the command parser. Set by COMVAL or block data in MENDEL
	ICHAR	Character under examination
	IPLUS	+ (all codes are right justified, 7 bit with zero fill)
	IMINUS	-
	ICOM	, May be manipulated to temporarily alter the argument separating character
	IUPARR	↑
	ISTAR	*
	ISPACE	*<blank>*
	IQUOTE	'
	IDQUOTE	"
	IPER	.
	ICR	*<cr>*
	IALT	*<altmode>*

Cont'd

APPENDIX C - PRTE COMMON BLOCKS

BLOCK NAME	VARIABLES	DESCRIPTION
COMTAB		Command Table and MNB file header information. The size of this common block is determined at load time by a loader code in the Jump Table. The first 64 words contain the MNB file header. The next 25Ø words contains the active menu. The remainder of the common block varies in size according to the number of global commands defined, but is a minimum of 25Ø additional words.
	PIGLET(2)	9-character, 5/7 ASCII MNB filename
	NODIR	Number of entries in the MAT
	NGCOM	Number of global commands defined
	MGSTRT	Starting block number of global commands
	NARGB	Block number of argument-getting menu (present, but facility unimplemented)
	MNSTRT	Block number of starting menu
	JMPDMP	Index in SNT of GAP dump procedure (unimplemented)
	INTDMP	Dumping interval in minutes (unimplemented)
	IDSZ	Display size (Ø or 1)
	LABRV	.TRUE. means abbreviated command mode
	IKDEV	Active keyboard code -1 PFA (unimplemented) Ø TTA 1 LTA 2 TEKTRONIX (unimplemented)
	LHARD	.TRUE. if hardcopy echo of keyboard device is desired
	LSDEV	Active stylus device code Ø Sparkpen 1 Lightpen 2 DMAC (unimplemented) 6 Tektronix (unimplemented)
	LNTERV	Wait interval after lightbutton hit in DO loop iterations. 86 iterations = 1 millisecond
	NOTREC	Total number of blocks in the MNB file
	MNUWRT	.TRUE. if the SVMNU option is to be obeyed. Set by the SAVE command
	NSBREC	Starting block number of the SNT
	FILOG(2)	Name of log file (unimplemented)
	FILERR(2)	Name of error message text file (unimplemented)
	FILHLP(2)	Name of help file (unimplemented)
	NSBBR	Number of entries in the SNT
	NBMNU	Number of MAT blocks available
	NBSBR	Number of SNT blocks available

Cont'd

APPENDIX C - PRTE COMMON BLOCKS

BLOCK NAME	VARIABLES	DESCRIPTION
	NBGLB	Number of global menu blocks available
	LBMNU	Last MAT block used
	LBSBR	Last SNT block used
	LBGBL	Last global menu block used
	LOGON	.TRUE. if commands are to be logged (unimplemented)
	IFIL(24)	Unused, words 41-64 of the MNB file header
	MHEAD(1Ø)	Command Table, current menu block header (see Appendix E)
		MHEAD(1) is the active command source code word reset by the DSABL command
		BIT 17 = 1 Keyboard active
		BIT 16 = 1 Lightbuttons active
		BIT 15 = 1 Pushbuttons active
		BIT 14 = 1 Real time clock active
		BIT 13 = 1 BSI active (umimplemented)
	MNU(24Ø)	Command Table, control, local, and pushbutton command nodes (see Appendix E)
		The current menu block is read from the MNB file into MHEAD and MNU.
		Character strings in the Command Table are displayed from the lightbutton files.
	MGLOBE(25Ø)	Command Table, first global menu block (see Appendix E). The size of MGLOBE is extended by the loader in blocks of 25Ø words according to the number of global commands defined in the GAP. All global menu blocks are read into core before the starting menu is entered.
DARG		Number argument buffer (see ARGTP)
	RARG(15)	Double precision array containing number argument values
	IRP	Index of next free entry in RARG
DISER		Error message string buffer
	ERRSTR(15)	Holds displayed error message area text
	MAXECH	Maximum number of characters allowed in buffer
	IXE,IYE	X-Y coordinates of start of error message display
	IXEØ,IYEØ	X-Y coordinates of start of error message display, size Ø
	IYE1,IYE1	X-Y coordinates of start of error message display, size 1

Cont'd

APPENDIX C - PRTE COMMON BLOCKS

BLOCK NAME	VARIABLES	DESCRIPTION
ERRCON		Error message flags, addresses, and buffers
	LERR	.TRUE. if an error occurred in the last command
	LFATL	.TRUE. if an ABORT error occurred in the last command
	LDISP	.TRUE. if errors are to be displayed (MENDEL only)
	LERPRT	.TRUE. if error messages are to be output to TTA
	LOGERR	.TRUE. if error messages are to be logged (not implemented)
	LARGS	.TRUE. if there are arguments to be retrieved and printed (not implemented)
	LEREST	<Control P> restart address for teletype (same as NRMRET)
	ERRNAM(2)	Name of command causing last error (not implemented)
	INTAK	.TRUE. if MENDEL 'I' option encountered
	NRMRET	Same as LEREST.
	LSTERR	Number of last error message which occurred.
	EMESS(9)	Error message teletype output buffer
	LABORT	MENDEL abort flag (not used)
HITBUT		Pushbutton and lightpen hit communication buffers
	LXX,LYY	X-Y coordinates of start of vector causing a lightpen hit
	NAMR	Name register value set during lightpen hit
	PBNEW(6)	Logical state of each pushbutton
	PBDUM(6)	Logical array used by PBHIT
	TPB	.TRUE. if a pushbutton hit occurred when lightpen hit was requested
	TLP	.TRUE. if a lightpen hit occurred when a pushbutton hit was requested
LAYDAT		Contains fixed text for 'MENU' and 'EXIT' display
	TMEN	'5HMENU' set by COMVAL
	RMEN	'5HEXIT' set by COMVAL
	LMEN	Y-coordinate of 'MENU' text
	LEXI	Y-coordinate of 'EXIT' text
MNUDAT		Contains current Command Table and menuing information
	IOREC	Old menu block number
	INREC	New menu block number
	NOTCOM	Total number of command nodes in the Command Table (including globals)
	MNUTP	Word index of last node in the Command Table
	LDIRB	Last MAT block used
	LGLOBF	Unused

Cont'd

APPENDIX C - PRTE COMMON BLOCKS

BLOCK NAME	VARIABLES	DESCRIPTION
OUTMOD		Keyboard input and teletype output, device and display control
	IHARD	Same as LHARD in COMTAB
	FILMON(2)	Name of log file (unused)
	KDEV	Active keyboard device code. Same as IKDEV in COMTAB
	IREST	*<Control P>* restart address for TTA..Same as NRMRET in ERRCON
	LXMARG,LYMARG	Starting X-Y coordinates of keyboard input string display
	MAXCH	Maximum number of characters allowed for keyboard input
	IABRV	Same as LABRV in COMTAB
	CBUF(2Ø)	Keyboard input buffer, displayed indirectly
	LXMARØ, LYMARØ	Starting X-Y coordinates of keyboard input string display, size Ø
	LXMAR1,LYMAR1	Starting X-Y coordinates of keyboard input string display, size 1
PAGMNU		MNB file I/O control
	IFSIZ	Number of blocks in MNB file
	FNAM(2)	5/7 ASCII name of MNB file
	ICREC	Number of last block accessed +1
	IRSZ	Block size, always 25Ø words
PIGDSP		PRTE display files
	LPIGDP(20)	PIGS display file
	LSYSDS(5Ø)	PRTE display file
	LPROM(15)	Prompting area display file
	LDISER(15)	Error message display file
	LKEYIN(16)	Keyboard input display file
	LDOT(9)	Tracking dot display file
	LLAY(32)	Display file for rectangular border and fixed text
	LTBUTS(18)	Combined lightbutton display file
	LTITLE(34)	Control area lightbutton display file
	LPBS(90)	Pushbutton lightbutton display file
	LLOCAL(23Ø)	Local lightbutton display file
	LGLOBL(9Ø)	Global lightbutton display file
	LSOFF1	Supplemental argument for blanking PRTE display (not used)
	LSOFF2	Supplemental argument for blanking PRTE display (not used)

Cont'd

APPENDIX C - PRTE COMMON BLOCKS

BLOCK NAME	VARIABLES	DESCRIPTION
PROMP		Prompting message control and buffers
	ARRYL(15)	Text buffer for prompting messages, displayed indirectly
	MAXPCH	Maximum number of characters allowed in text buffer
	IXP,IYP	Starting coordinates of prompting message display,
	IXPØ,IYPØ	Starting coordinates of prompting message display, size Ø
	IXP1,IYP1	Starting coordinates of prompting message display, size 1
PUSHB		Pushbutton hits and state
	IBUTNO	Button number of last pushbutton hit
	PBOLD(6)	Logical state of the 6 pushbuttons
SCHARR		Clock schedule
	ISCH(46)	Clock scheduling array, 9 words/entry
		Word 1-4 9 character command name <altmode> or Ø = end of schedule -1 = garbage entry, descheduled
		5 Number of selections before command is descheduled -1 means repeat indefinitely
		6 Interval in seconds between selections
		7 Interval in clock pulses between selections
		8 Next due time, seconds
		9 Next due time, pulses
STYLS		Active stylus device control
	ISDEV	Active stylus device code, identical to LSDEV in COMTAB
	INTERV	Same as LNTERV in COMTAB
	NAMREG	Name register value of last lightbutton hit
	ISIZ	Same as IDSZ in COMTAB
TIMEB		Clock command polling and control
	ISEC	Current time in seconds
	IPULSE	Current time in pulses
	IOFF	Ø Clock running 1 Clock stopped
	ISCHP	Index in clock schedule array of start of next search for a due command
	LSTH(4)	Unused

APPENDIX C - MENDEL COMMON BLOCKS

ARGTP, CARG, CHARS, COMTAB, DARG, ERRCON, and PAGMNU are identical in size and function to labelled common blocks of the same name in PRTE.

BLOCK NAME	VARIABLES	DESCRIPTION
CODBIN		Binary relocateable output buffer control for Jump Table
	ICDWD	Index in ICOD (below) of current loader code word being formed
	ICDX	Bit position of next six-bit loader code in current code word
	IDATWD	Index in ICOD (below) of current loader data word
	IBFLG	Unused
	IBUF(2)	Header word pair for output buffer
	ICOD(24)	Binary output buffer for one loader record
MDLBUF		Header, menu block, MAT, and SNT buffers for the editor-assembler
	IHEAD(64)	Indentical to first 64 words of COMTAB in PRTE
	MBUF(25Ø)	Holds current menu block under definition
	DMNU(125)	Holds one block of the MAT
	DSBR(125)	Holds one block of the SNT
MDLVAR		Control variables and pointers for the editor-assembler
	ITRP	Current MENDEL context, some value between Ø and 6
	IED	.TRUE. if in EDIT mode, .FALSE. if in CREAT mode
	IPC	Word address of current command node, minus 1Ø
	IBLK	Current block number under definition
	IPM	Word address, in the current block, of the last MAT entry defined
	IPS	Word address, in the current block, of the last SNT entry defined
	FNAME(2)	MNB filename under definition
	APLNAM(2)	GAP name
	MNUREC	Block address of last menu defined in the MAT
	NTMNU	Last available MAT block
	NTSBR	Last available SNT block
	NTGLB	Number of last global block available
	LASTB	Last global or local menu block filled
	LIST	.TRUE. if listing of MENDEL source being produced
	FLIST(2)	Name of list file

APPENDIX C - .LIBRP COMMON BLOCKS

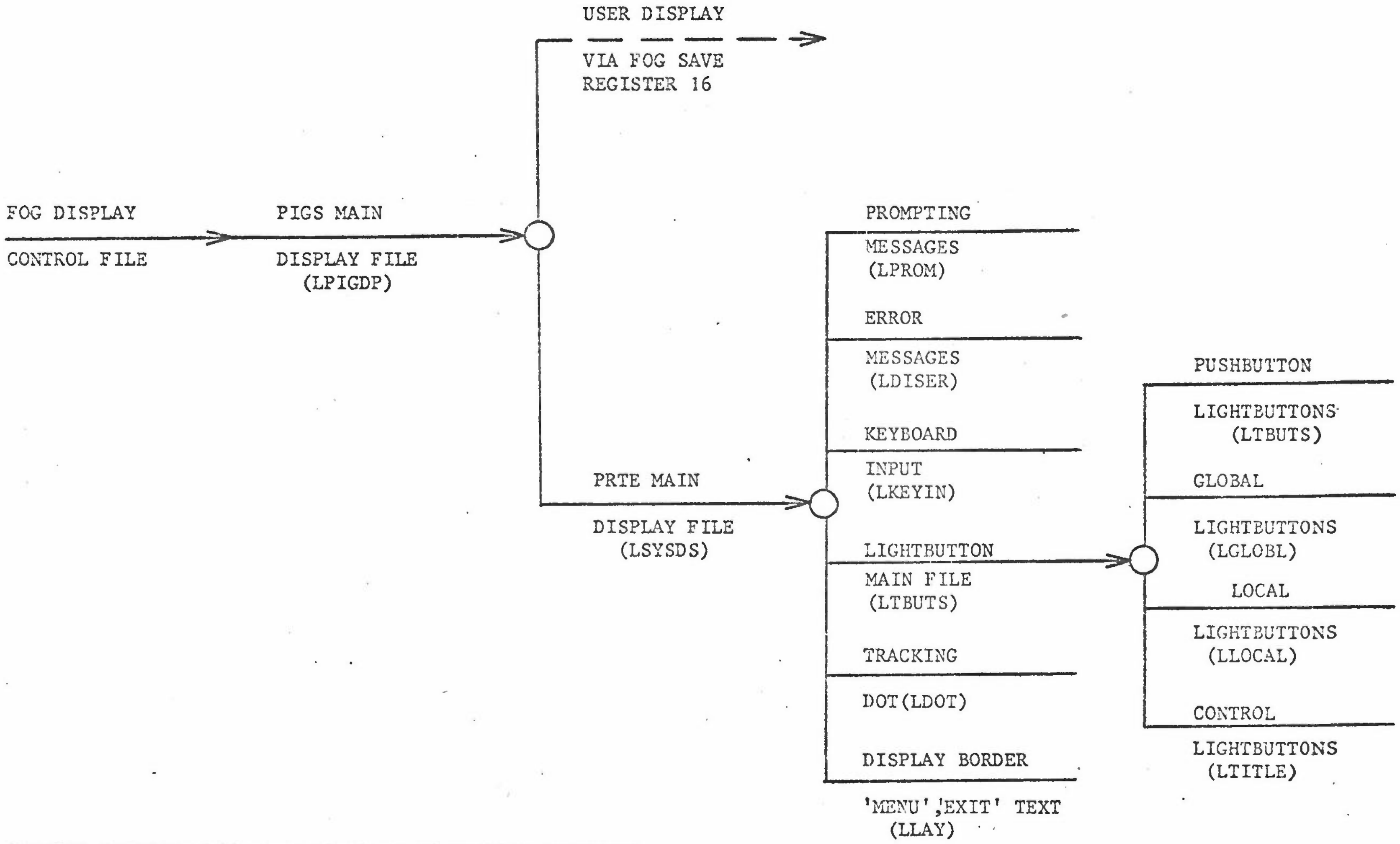
Labelled common blocks GRIDAT, PSHBUT, and STYDAT are used by subroutines GRID and STYLI and are fully described in the PDP15 FORTRAN LIBRARY.

APPENDIX C - MNB FILE BLOCK POINTERS

The following table is included to clarify the use of pointers and variables by MENDEL and PRTE in accessing MNB files. The variables are located in common blocks COMTAB,MDLBUF, and MDLVAR, previously discussed.

USE BLOCK TYPE	First Block	Number of last block available	Number of blocks available	Last block used	Number of Entries
HEADER	1	1	1	1	40
MAT	2	NTMNU	NBMNU	LBMNU	NODIR
SNT	NSBREC	NTSBR	NBSBR	LBSBR	NSBBR
GLOBL	MGSTRT	NTGLB	NBGBL	LBGBL	NGCOM
ARGET	NARGB	NARGB	1	NARGB	Ø
MENU	MNSTRT	NOTREC	NOTREC-NARGB	LASTB	MBUF(2)

APPENDIX D - PIGS DISPLAY FILE STRUCTURE*



*ARROWS INDICATE FOG DRAW COMMANDS TO DISPLAY SUBFILES.
ARRAY NAMES OF DISPLAY FILES ARE GIVEN IN BRACKETS.

APPENDIX D - PIGS DISPLAY FILE STRUCTURE, LIGHTBUTTON FILES

Subroutines CBUTHT and BUTHIT are used by PRTE to detect lightbutton hits by the active stylus device, either the lightpen or sparkpen. In order for these routines to work properly, lightbutton display files must follow a rigid format.

Subroutine BUTHIT detects lightbutton hits with the lightpen using LPHIT, which simply returns a unique name register value set up in the display file. If the sparkpen is the active device, BUTHIT compares setpoint information in the display file with the stylus position to determine if a hit has occurred. If so, it retrieves the proper name register value from the SKIP2 instruction in the file itself. Lightbuttons are winked by PRTE using a PARAM2 instruction associated with each button.

Subroutine BUTDIS can be used to create lightbutton files with the required format for hit detection using CBUTHT or BUTHIT. *Simple* lightbutton files, as they are called, display columns or rows of names of variable scale. Lightbuttons may be displayed in the offset or main display area and spacing between buttons is variable. Simple lightbutton files may also be created using MACRO-15, as the example below illustrates.

Example 1

PRTE array LTITLE contains the control area lightbutton display file. The two lightbuttons it contains appear on the display opposite the text strings 'MENU' and 'EXIT'. Each button displayed requires 14 instruction words in the file. An extra 5 words are required to make a well-formed FOG display file. The control area lightbutton file could be coded in MACRO-15 as follows (PRTE uses BUTDIS):

Simple Lightbutton File - LTITLE

```
.EBREL                          /Use 13-bit addresses
CHARS =  060000                 /Character string instruction
DNOP  =  200000                 /Display NOP
DJMP  =  600000                 /Display NOP instruction
PX    =  144000                 /Position beam instruction, x direction
PY    =  140000                 /Position beam instruction, y direction
OFFSET = 1                      /Use offset area
ISCALE = 1                      /Large text for buttons
MENU          .ASCII   'PATH'<175>    /Button 1
MENDAT        .ASCII   'INIT'<175>    /Data for button 1
EXIT          .ASCII   'BACKG'<175>   /Button 2
EXITD         .ASCII   <175>          /Data for button 2
XB1 = 12                        /X-Y coordinates of button 1
YB1 = 1654                      /in offset area
XD1 = XB1+1Ø6                   /X-Y coordinates of data area 1
YD1 = YB1                       /in offset area
XB2 = 12                        /X-Y coordinates of button 2
YB2 = 1524                      /below button 1
```

```
XD2 = XB2+1Ø6           /X-Y coordinates of data area 2
YD2 = YB2               /(All of the coordinates above would be
                        /automatically computed by BUTDIS)
LTITLE      41          /Length of display file = NOBUTTON*14+5
             Ø          /Return address planted here
            DNOP        /FOG blanking word
/Button 1   2344ØØ+1&177 /SKIP2 - load 1 into name register
            220004       /PARAM3 instruction
            211056+OFFSET & 1 /PARAM2 - used to blink a
                              /button, enable lightpen, and
                              /select offset area
            203020+ISCALE & 17 /PARAM1 - set scale 1 chars
            PY!YB1             /Position beam for first
            PX!XB1             /button name
            CHARS*    .+2      /Display button name, indirect
            DJMP      .+2      /Avoid indirect address
            .DSA      MENU     /Address of name.  PRTE would
                               /point to word 1 of command
                               /node 1
            PY!YD1             /Position beam for first
            PX!XD1             /data area.  Y value the same
            CHARS*    .+2      /Display button 1 data area
            DJMP      .+2      /Avoid indirect address
            .DSA      MENDAT   /Address of data area.  PRTE display
                               /would point to word 4 of
                               /command node 1
/Button 2 essentially repeats the previous 14 words
            2344ØØ+1&177
            220004
            211056+OFFSET&1
            203020+ISCALE&17
            PY!YB2
            PX!XB2
            CHARS* .+2
            DJMP   .+2
            .DSA   EXIT
            PY!YD2
            PX!XD2
            CHARS* .+2
            DJMP   .+2
            .DSA   EXITD
/Finish button 2
            211056 /PARAM2-turn blink off
            DJMP* LTITLE+2 /Return
            .DBREL /Back to 12-bit addresses
            .END
```

Subroutine CBUTHT may be used to check for hits on more than one simple lightbutton file at a time. In order use CBUTHT, a *combined* lightbutton display file must be constructed, using FOG or MACRO-15, which contains only DRAW's to simple files. CBUTHT scans the combined file in order to discover the address of each simple file, then calls BUTHIT.

Example 2

PRTE display file LTBUTS contains the simple lightbutton files LPBS,LGLOBL, LPBS,LGLOBL,LLOCAL, and LTITLE. This combined lightbutton display file could be constructed using FOG or MACRO-15, as shown below.

Using FOG from FORTRAN:

```
     DIMENSION  LTBUTS(16)
     LTBUTS(1)=Ø
     CALL DCHOOS (LTBUTS,1)
     CALL DRAW (Ø,LPBS(1))
     CALL DRAW (Ø,LGLOBL(1))
     CALL DRAW (Ø,LLOCAL(1))
     CALL DRAW (Ø,LTITLE(1))
```

Using MACRO-15 with the same VT15 instruction definitions as Example 1

```
     .EBREL
     DJMS=64ØØØØ
LTBUTS          17              /Display file length
                 Ø              /Return address
                DNOP            /Fog blanking word
                DJMP   .+2      /DRAW LPBS
                .DSA LPBS
                DJMS* .-1
/               DJMP   .+2      /DRAW LGLOBL
                .DSA LGLOBL
                DJMS* .-1
/
                DJMP   .+2      /DRAW LLOCAL
                .DSA LLOCAL
                DJMS* .-1
/
                DJMP   .+2      /DRAW LTITLE
                .DSA LTITLE
                DJMS* .-1
/
                DJMP* LTBUTS+1  /RETURN
                .DBREL
                .END
```

APPENDIX E MNB FILE STRUCTURE

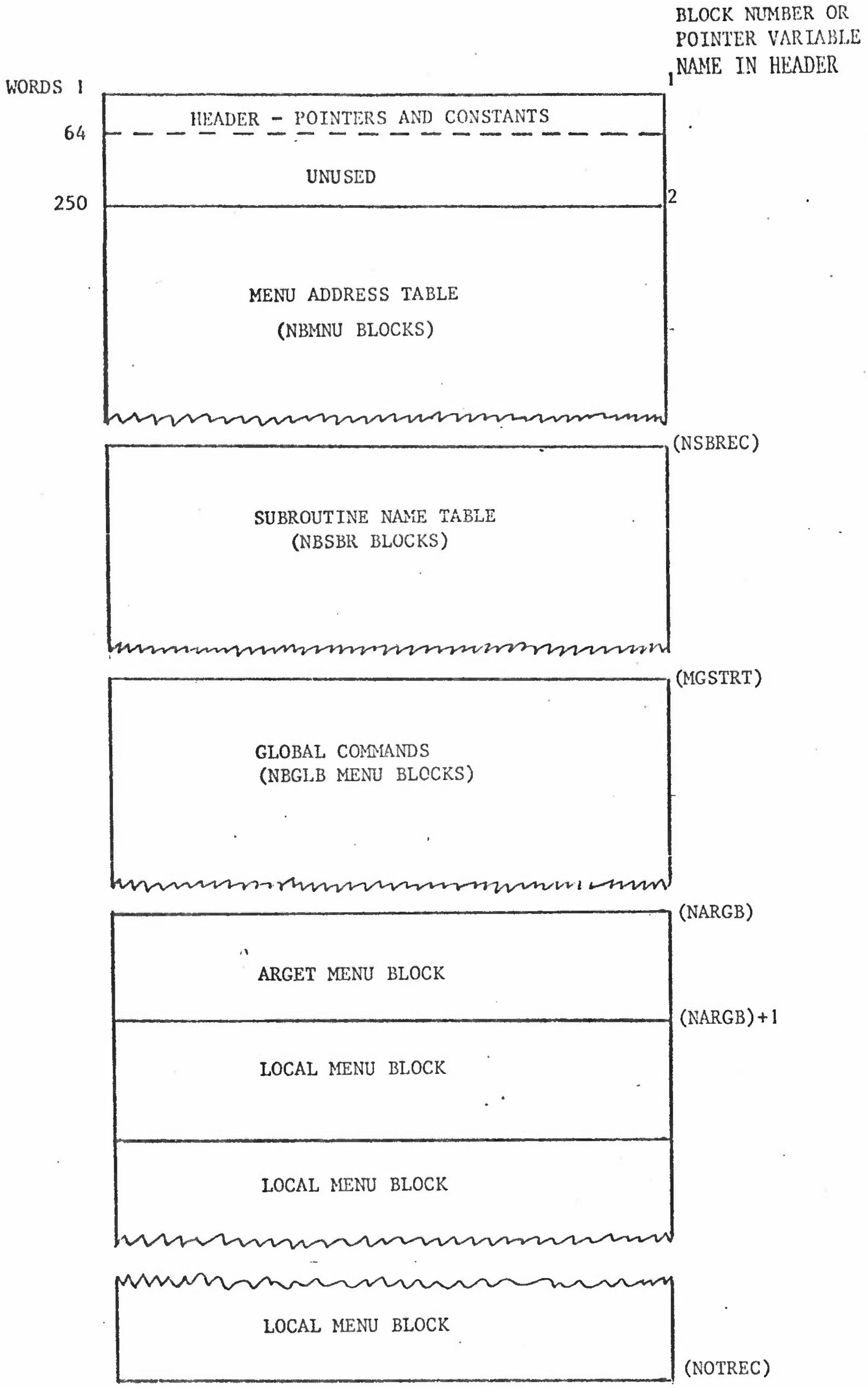
The following tables and illustrations explain the MNB file, menu block, and command node structures as created by the MENDEL editor-assembler and interpreted by PRTE. The disc file format of these structures is related to MENDEL and PRTE common blocks as described in Appendix C. The symbols appearing next to the MNB file blocks are COMTAB labelled common block variables.

The first illustration shows the layout of a well-formed binary MNB file. Subsequent pages describe the format and function of each of the 250-word disc blocks which make up such a file.

E

MNB FILE STRUCTURE

Each block contains 250 binary words



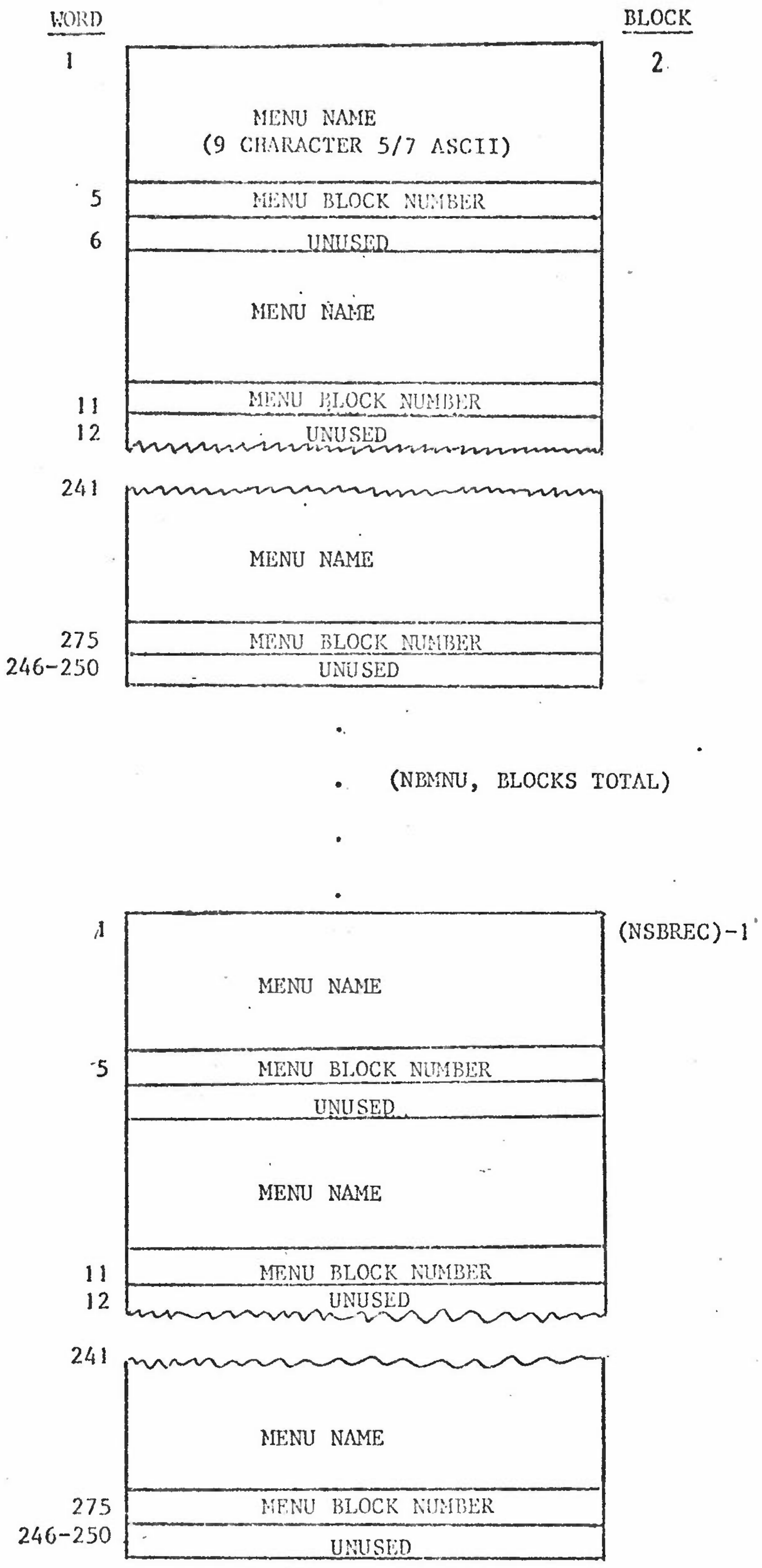
MNB File Header Entries

Only words 1-64 are read into the COMTAB common block in core. Words 41-250 are currently unused and are set to Ø. The function of each entry is included in the discussion of COMTAB in Appendix C.

HEADER WORD	COMTAB VARIABLE
1-4	PIGLET(2)
5	NODIR
6	NGCOM
7	MGSTRT
8	NARGB
9	MNSTRT
10	JMPDMP
11	INTDMP
12	IDSZ
13	LABRV
14	IKDEV
15	LHARD
16	LSDEV
17	LNTERV
18	NOTREC
19	MNUWRT
20	NSBREC
21-24	FILOG(2)
25-28	FILERR(2)
29-32	FILHLP(2)
33	NSBBR
34	NBMNU
35	NBSBR
36	NBGLB
37	LBMNU
38	LBSBR
39	LBGBL
40	LOGON
41-64	IFIL(24) Present in COMTAB but unused
65-250	Unused and not in COMTAB

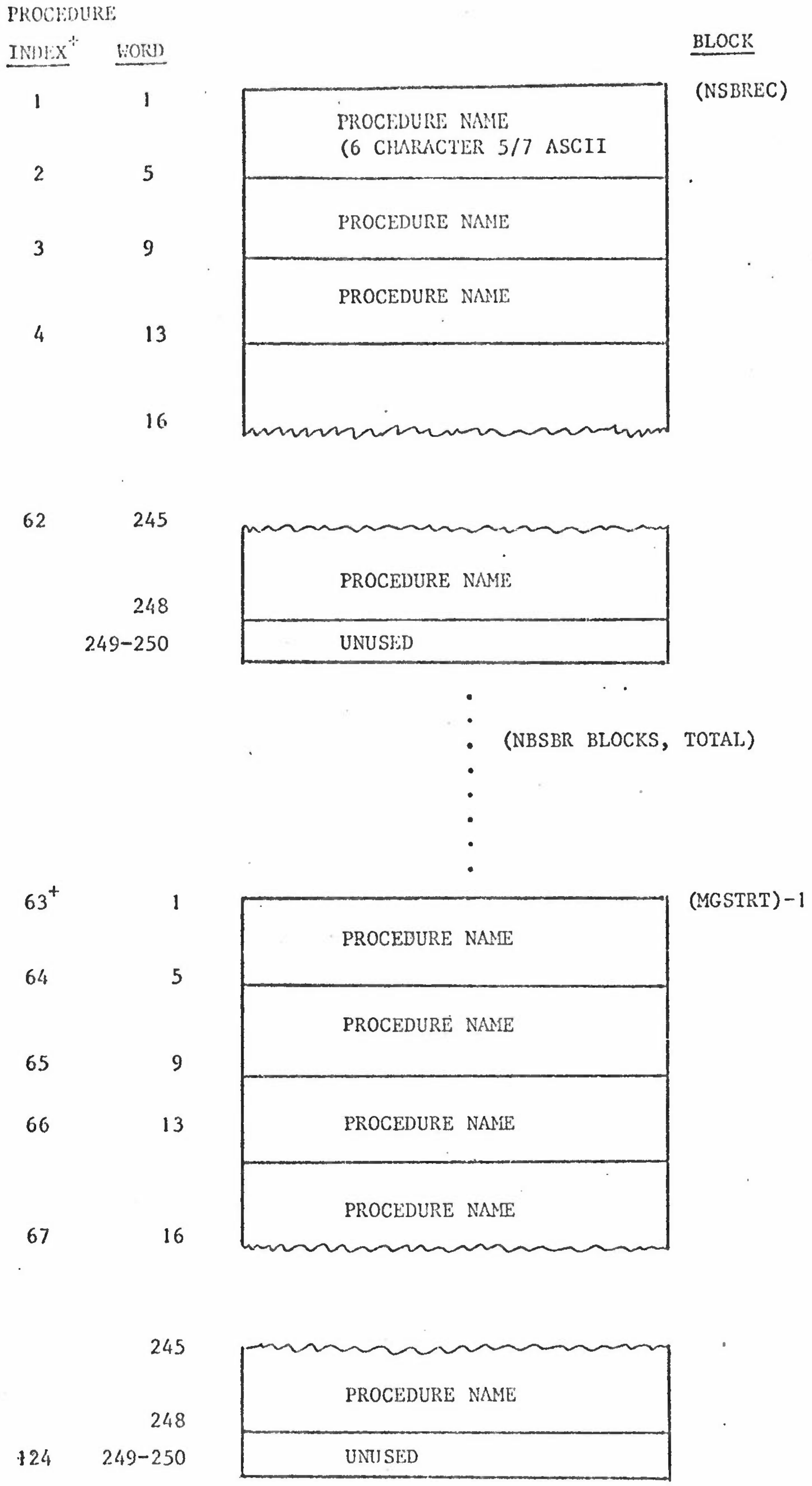
E

MENU Address Table Format*



*ALL NUMBERS ARE DECIMAL RADIX

Subroutine Name Table Format*   *



PROCEDURE INDEX+	WORD		BLOCK
1	1	PROCEDURE NAME (6 CHARACTER 5/7 ASCII	(NSBREC)
2	5	PROCEDURE NAME	
3	9	PROCEDURE NAME	
4	13		
	16		
62	245	PROCEDURE NAME	
	248		
	249-250	UNUSED	
		(NBSBR BLOCKS, TOTAL)	
63+	1	PROCEDURE NAME	(MGSTRT)-1
64	5	PROCEDURE NAME	
65	9	PROCEDURE NAME	
66	13	PROCEDURE NAME	
67	16		
	245	PROCEDURE NAME	
	248		
124	249-250	UNUSED	

*ALL NUMBERS ARE DECIMAL RADIX

+PROCEDURE INDICES ARE USED IN COMMAND NODES. INDEXING IS CONSECUTIVE ACROSS BLOCK BOUNDARIES. THIS INDEXING ASSUMES NBSBR=2

E

MENU Block Format*

FUNCTION (GLOBAL CONTEXT)	STARTING WORD OF NODE	LAST WORD OF NODE	FUNCTION (MENU CONTEXT)
MENU HEADER	1	10	MENU HEADER
GLOBAL 1 (DISPLAYED)	11	20	ENTER (COMMAND NODES) ↓
2	21	30	EXIT
3	31	40	PUSHBUTTON 1
4	41	50	PUSHBUTTON 2
5	51	60	PUSHBUTTON 3
6	61	70	PUSHBUTTON 4
GLOBAL 7 (NON-DISPLAYED)	71	80	PUSHBUTTON 5
8	81	90	PUSHBUTTON 6
9	91	100	LOCAL 1
10	101	110	LOCAL 2
11	111	120	LOCAL 3
12	121	130	LOCAL 4
13	131	140	LOCAL 5
14	141	150	LOCAL 6
15	151	160	LOCAL 7
16	161	170	LOCAL 8
17	171	180	LOCAL 9
18	181	190	LOCAL 10
19	191	200	LOCAL 11
20	201	210	LOCAL 12
21	211	220	LOCAL 13
22	221	230	LOCAL 14
23	231	240	LOCAL 15
24	241	250	LOCAL 16

GLOBAL COMMAND NODES
MAY CONTINUE WITH MORE MENU BLOCKS
OF IDENTICAL FORMAT. ALL BLOCKS HAVE HEADERS

*ALL NUMBERS ARE DECIMAL RADIX

WORD	MENU HEADER FORMAT					
BIT		13	14	15	16	17
1	ACTIVE COMMAND SOURCE WORD FOR MENUS	BSI	CLOCK	PUSHB	LTBUT	KEYBD
2	NUMBER (1-24) OF LAST DEFINED COMMAND NODE IN THIS MENU BLOCK					
3	*SVMNU* FLAG. IF .TRUE. AND *SAVE* OPTION .TRUE., WRITE THIS MENU TO DISC					
4.1Ø	UNUSED					

WORD	COMMAND NODE FORMAT			
1-4	COMMAND NAME, 5 OR 9-CHARACTER 5/7 ASCII *<ALTMODE>* IN CHARACTER 6 OR 10. IF COMMAND IS INACTIVE, CHARACTER 1Ø CONTAINS NORMAL CHARACTER 1,AND CHARACTER 1 IS *<ALTMODE>*			
5-8	DATA AREA. FORMAT IS IDENTICAL TO COMMAND NAME. BOTH COMMAND NAMES AND DATA AREAS ARE DISPLAYED FROM LIGHTBUTTON FILES USING THE CHARACTER STRING INDIRECT INSTRUCTION.			
BIT	Ø	1-5	6	7-17
9	1 COMMAND ACTIVE Ø COMMAND INACTIVE	MAXIMUM NUMBER OF ARGUMENTS. -1 MEANS ANY NUMBER OF ARGUMENTS	1 NO DELAY Ø NORMAL DELAY BEFORE PROCEDURE EXECUTION	INDEX OF PROCEDURE TO BE EXECUTED IN SNT. Ø MEANS NONE.
BIT	Ø	1	7-17	
1Ø	1 EXECUTE EXIT PROCEDURE BEFORE MENU CHANGE Ø DO NOT EXECUTE EXIT PROCEDURE	1 EXECUTE ENTRY PROCEDURE BEFORE MENU CHANGE Ø DO NOT EXECUTE ENTRY PROCEDURE	MENU BLOCK NUMBER OF NEW MENU TO BE ACTIVATED Ø MEANS NO NEW MENU.	

APPENDIX F - OVERLAY DESCRIPTIONS AND LOAD MAPS

PRTE

```
CHAIN ACL

NAME XCT FILE
>PRTE
LIST OPTIONS & PARAMETERS
>SZ,PAR,PAL,XSP,VIC/PIGDSP,DISER,PROMP,LAYDAT,COMTAB,OUTMOD/
DEFINE RESIDENT CODE
>GAP
DESCRIBE LINKS & STRUCTURE
>LK1=#COMVAL
>LK2=#YYSTRT,#OPMNB,#RDMNU,#WTMNU
>LK3=#DISINT
>LK4=#POLL
>LK5=#RESOLV
>LK1:LK2:LK3:LK4:LK5
>
LINK TABLE
        77505-77636 00132

RESIDENT CODE
GAP     77502-77504 00003
PIGS2   76502-77501 01000
RESLIB  76471-76501 00011
YYFLA   76432-76470 00037
CLERR2  76323-76431 00107
NMNU2   76135-76322 00166
JUMPT2  76111-76134 00024
CLPRM   76056-76110 00033
WINK    75662-76055 00174
TIMEP   75606-75661 00054
TABLET  75530-75605 00056
FOG     73501-75527 02027
VTERR2  73461-73500 00020
FRMS4   73301-73460 00160
YYPRLN  73203-73300 00076
DISERR  73140-73202 00043
MESDIS  73070-73137 00050
CVICH   72655-73067 00213
JPSTAR  72644-72654 00011
IBITS   72611-72643 00033
MSTR    72536-72610 00053
MVC     72400-72535 00136
LTA.    71035-72377 01343
.PFDUM  71033-71034 00002
PRIN1   70775-71032 00036
LTORPB  70663-70774 00112
DEFINE  66650-67777 01130
ADJ1    70644-70662 00017
.DA     70566-70643 00056
.SS     70477-70565 00067
STOP    70464-70476 00013
SPMSG   70345-70463 00117
.FLIB   70057-70344 00266
FIOPS   65447-66647 01201
INTFAE  65316-65446 00131
RELFAE  64217-65315 01077
OTSER   64007-64216 00210
.CB     70035-70056 00022
COMTAB  62723-64006 01064
OUTMOD  62634-62722 00067
PIGDSP  61457-62633 01155
MNUDAT  70027-70034 00006
ARGTP   61416-61456 00041
STYLS   70023-70026 00004
ERRCON  61355-61415 00041
DISER   61310-61354 00045
PACMNU  70014-70022 00007
LAYDAT  70006-70013 00006
PUSHB   61301-61307 00007
HITDET  61260-61300 00021
CARG    61221-61257 00037
DARG    61143-61220 00056
TIMER   61133-61142 00010
CHARS   61117-61132 00014
SCHALA  61041-61116 00056
PROMP   60774-61040 00045
```

F

```
LINK -- LK1
COMVAL 60377-60773 00375

LINK -- LK2
YYSTRT 60670-60773 00104
PIGFIL 60613-60667 00055
COMINT 57200-57777 00600
OPMNB2 60440-60612 00153
RDMNU2 60346-60437 00072
WTMNU2 60260-60345 00066
FNAME  56674-57177 00304
RBINIO 60145-60257 00113
PANCOM 56170-56673 00504
FILE   55575-56167 00373
BINIO  55146-55574 00427

LINK -- LK3
DISINT 60331-60773 00443
BUTINT 60034-60330 00275
SYSDN2 57472-57777 00306
PIGDNT 57405-57471 00065
LAYOU2 57222-57404 00163
PROMPI 57140-57221 00062
DISERI 57056-57137 00062
MESINT 56727-57055 00127
BUTDIS 56132-56726 00575
CLOCK  56075-56131 00035
KEYST3 55730-56074 00145
KEYS   55554-55727 00154
DOTINT 55472-55553 00062
FOG4   55255-55471 00215

LINK -- LK4
POLL3  60371-60773 00403
CBUTHT 60275-60370 00074
BUTHT2 57203-57777 00575
PBHIT  60071-60274 00204
LPHIT  57041-57202 00142
CLOCHK 56432-57040 00407
DSCHED 56172-56431 00240
KEYIN3 55603-56171 00367
KEYS   55427-55602 00154
DOTMV2 55274-55426 00133
GOTO   60043-60070 00026

LINK -- LK5
RESLV2 60314-60773 00460
GETCH2 60025-60313 00267
TARGS2 57603-57777 00175
PARG   57353-57602 00230
PNUMB  57021-57352 00332
PSTRNG 56547-57020 00252
PINT   56240-56546 00307
TERMIN 56136-56237 00102
NXTCHR 56111-56135 00025
BLANKS 56040-56110 00051
.BD    55706-56037 00132
.BH    55652-55705 00034
.DE    55551-55651 00101
.DF    55412-55550 00137
.DC    55343-55411 00047
GOTO   55315-55342 00026
DOUBLE 55112-55314 00203

CORE REC'D
       55112-77636 22525
```

PATH

```
CHAIN ACL

NAME XCT FILE
>PATH
LIST OPTIONS & PARAMETERS
>SZ,PAR,PAL,XSP,
-VTC/PIGDSP,DISFR,PROMP,LAYDAT,COMTAB,OUTMOD,BACKG,DATB,PTHDAT/
DEFINE RESIDENT CODE
>PATH,CPATH
DESCRIBE LINKS & STRUCTURE
>LK1=#COMVAL
>LK2=#YYSTRT,#OPMNB,#RDMNU,#WTMNU
>LK3=#DISINT,#CLOCK
>LK4=#POLL,#CLOCHK,#DSCHED
>LK5=#RESOLV
>LK1:LK2:LK3:LK4:LK5
>
LINK TABLE
        77444-77636 ØØ173

RESIDENT CODE
PATH    77424-77443 ØØØ2Ø
PINIT   77415-77423 ØØØØ7
STYDT2  77375-77414 ØØØ2Ø
DATACL  77226-77374 ØØ147
BLKDIS  77162-77225 ØØØ44
INITD2  76735-77161 ØØ225
PAGECL  7664Ø-76734 ØØØ75
DRAWC   76374-76637 ØØ244
REDRAW  76223-76373 ØØ151
CEL     76Ø22-76222 ØØ2Ø1
DRAWB   75711-76Ø21 ØØ111
BACKGR  75567-7571Ø ØØ122
PLAYBA  75Ø17-75566 ØØ55Ø
SHOW    74614-75Ø16 ØØ2Ø3
FRATE   74613-74613 ØØØØ1
GRIDAT  746ØØ-74612 ØØØ13
SHOWIT  74424-74577 ØØ154
LTPEN1  74Ø2Ø-74423 ØØ4Ø4
PIGS2   73Ø2Ø-74Ø17 Ø1ØØØ
RESLIB  73ØØ7-73Ø17 ØØØ11
YYFLA   7275Ø-73ØØ6 ØØØ37
CLEAR2  72641-72747 ØØ1Ø7
NMNU2   72453-7264Ø ØØ166
JUMPT2  72427-72452 ØØØ24
USFKEY  722Ø2-72426 ØØ225
USFSTY  71765-722Ø1 ØØ215
USFGRI  71612-71764 ØØ153
WDAT    71562-71611 ØØØ3Ø
STYLI2  65633-67777 Ø2145
PROMPT  71522-71561 ØØØ4Ø
CLPRM   71467-71521 ØØØ33
GRID    706ØØ-71466 ØØ667
WMNU    7Ø46Ø-7Ø577 ØØ12Ø
SKEDL   7Ø245-7Ø457 ØØ213
DSKFD   7Ø1Ø1-7Ø244 ØØ144
```

F

```
GETSI2  65523-65632 ØØ11Ø
GETDP2  65333-65522 ØØ17Ø
GETCH2  65Ø44-65332 ØØ267
LINK    6465Ø-65Ø43 ØØ174
SKEDUL  7ØØ44-7Ø1ØØ ØØØ35
DSKEDU  7ØØ22-7ØØ43 ØØØ22
SCHED   64366-64647 ØØ262
TIMEP   64312-64365 ØØØ54
KEYS    64136-64311 ØØ154
DOTMV2  64ØØ3-64135 ØØ133
TABLET  63725-64ØØ2 ØØØ56
ACTCM   63542-63724 ØØ163
DACTCM  63356-63541 ØØ164
CMFIND  63Ø43-63355 ØØ313
ERP     63Ø23-63Ø42 ØØØ2Ø
QUIT    63Ø14-63Ø22 ØØØØ7
FOG     6Ø765-63Ø13 Ø2Ø27
FOG2    6Ø674-6Ø764 ØØØ71
FOG4    6Ø457-6Ø673 ØØ215
VTERR2  6Ø437-6Ø456 ØØØ2Ø
ERRMS4  6Ø257-6Ø436 ØØ16Ø
YYPRLN  6Ø161-6Ø256 ØØØ76
DISERR  6Ø116-6Ø16Ø ØØØ43
MESDIS  6ØØ46-6Ø115 ØØØ5Ø
CVICH   57565-57777 ØØ213
JPSTAR  6ØØ35-6ØØ45 ØØØ11
INBITS  575Ø4-57564 ØØØ61
IBITS   57451-575Ø3 ØØØ33
MSTR    57376-5745Ø ØØØ53
MVC     5724Ø-57375 ØØ136
LTA.    55675-57237 Ø1343
.PFDUM  7ØØ2Ø-7ØØ21 ØØØØ2
PRIN1   55637-55674 ØØØ36
LTORPB  55525-55636 ØØ112
DEFINE  54375-55524 Ø113Ø
EDCODE  5412Ø-54374 ØØ255
ADJ1    541Ø1-54117 ØØØ17
IABS    6ØØ21-6ØØ34 ØØØ14
.DA     54Ø23-541ØØ ØØØ56
BCDIO   5ØØ43-54Ø22 Ø376Ø
.SS     47711-47777 ØØØ67
GOTO    47663-4771Ø ØØØ26
STOP    5ØØ3Ø-5ØØ42 ØØØ13
SPMSG   47544-47662 ØØ117
.FLTB   47256-47543 ØØ266
FIOPS   46Ø55-47255 Ø12Ø1
INTEAE  45724-46Ø54 ØØ131
DOUBLE  45521-45723 ØØ2Ø3
RFLEAE  44422-4552Ø Ø1Ø77
OTSER   44212-44421 ØØ21Ø
.CB     4417Ø-44211 ØØØ22
COMTAB  43Ø72-44167 Ø1Ø76
PTHDAT  33664-37777 Ø4114
PACKG   42472-43Ø71 ØØ4ØØ
DATB    417Ø3-42471 ØØ567
ARGTP   41642-417Ø2 ØØØ41
TIMEB   7ØØØ5-7ØØ17 ØØØ13
OUTMOD  41553-41641 ØØØ67
PIGDSP  4Ø376-41552 Ø1155
MNUDAT  6ØØ13-6ØØ2Ø ØØØØ6
STYLS   7ØØØ1-7ØØØ4 ØØØØ4
ERRCON  4Ø335-4Ø375 ØØØ41
DISER   4Ø27Ø-4Ø334 ØØØ45
```

F

```
PSCMNU 60004-60012 00007
LAYDAT 50022-50027 00006
PUSHB  50013-50021 00007
HITBUT 40247-40267 00021
CARG   40210-40246 00037
DARG   40132-40207 00056
CHARS  40116-40131 00014
SCHARR 40040-40115 00056
PSHBUT 50005-50012 00006
PROMP  33617-33663 00045

LINK -- LK1
COMVAL 33222-33616 00375

LINK -- LK2
YYSTRT 33513-33616 00104
PIGFIL 33436-33512 00055
COMINT 32636-33435 00600
OPMNU2 32463-32635 00153
RDMNU2 32371-32462 00072
WTMNU2 32303-32370 00066
FNAME  31777-32302 00304
RBINIO 31664-31776 00113
RANCOM 31160-31663 00504
FILE   30565-31157 00373
BINIO  30136-30564 00427

LINK -- LK3
DISINT 33154-33616 00443
BUTINT 32657-33153 00275
SYSDN2 32351-32656 00306
PIGDNT 32264-32350 00065
LAYOU2 32101-32263 00163
PROMPI 32017-32100 00062
DISERI 31735-32016 00062
MESINT 31606-31734 00127
BUTDIS 31011-31605 00575
CLOCK  30754-31010 00035
KEYST3 30607-30753 00145
DOTINT 30525-30606 00062

LINK -- LK4
POLL3  33214-33616 00403
CBUTHT 33120-33213 00074
BUTHT2 32323-33117 00575
PBHIT  32117-32322 00204
LPHIT  31755-32116 00142
CLOCHK 31346-31754 00407
DSCHED 31106-31345 00240
KEYIN3 30517-31105 00367

LINK -- LK5
RESLV2 33137-33616 00460
TARGS2 32742-33136 00175
PARG   32512-32741 00230
PNUMB  32160-32511 00332
PSTRNG 31706-32157 00252
PINT   31377-31705 00307
TERMIN 31275-31376 00102
NXTCHR 31250-31274 00025
BLANKS 31177-31247 00051
.PD    31045-31176 00132
.PH    31011-31044 00034
.DE    30710-31010 00101
.DF    30551-30707 00137
.DC    30502-30550 00047

CORE REQ'D
       30136-77636 47501
```

APPENDIX G - PRTE AND MENDEL COMMAND SYNTAX

The following is a BNF grammar specifying <command> as the sentence. It does not represent the fact that the character 9 is now allowed in an octal number. A maximum of 15 arguments, including the command name are allowed. Examples of valid input is provided in the next section.

```
<DIGIT>::=Ø/1/2/3/4/5/6/7/8/9
<ALPHA>::=A/B/C/D/.../X/Y/Z
<CHAR>::=<ANY ASCII CHARACTER>
<BLANKS>::=BLANK/BLANK<BLANKS>
<NONTERM>::=<ANY ASCII CHARACTER EXCEPT  BLANK OR , OR CR
        OR ALTMODE>
<NTAIL>::=<NONTERM>/<NONTERM><NTAIL>
<CSTRING>::=<ALPHA>/<ALPHA><NTAIL>
<STAIL>::=<CHAR>/<CHAR><STAIL>
<STRING>::=<CSTRING>/ ' <STAIL> ' / " <STAIL> "
<INTP>::=<BLANKS><DIGIT>/<BLANKS><DIGIT><INTP>
<FRACTION>::= . / . <INTP>
<U-MANTISSA>::=<INTP>/<INTP><FRACTION>/<FRACTION>
<MANTISSA>::= + <U-MANTISSA>/ - <U-MANTISSA>/<U-MANTISSA>

<S-INTP>::=<INTP>/ + <INTP>/ - <INTP>
<EXPONENT>::= ↑ <BLANKS><S-INTP>
<DNUMBER>::=<MANTISSA>/<MANTISSA><EXPONENT>

<NUMBER>: <DNUMBER>/ # <BLANKS><DNUMBER>
<OMIT>::=<BLANKS>,/,
<LEAVE>::=<BLANKS> * <BLANKS>
<ARG>::=<OMIT>/<LEAVE>/<STRING>/<NUMBER>
<ARGSTRING>::= , <BLANKS><ARG>/<BLANKS><ARG><ARGSTRING>
<ENDCOM>::= CR / ALTMODE
<COMMAND>::=<BLANKS><STRING><BLANKS><ARGSTRING><ENDCOM>
```

APPENDIX H - RESERVED PRTE SYMBOLS

In addition to the FOG subroutined names, certain global symbols are reserved by PRTE and should not be duplicated by the GAP as:

(1) Entry point, procedure, block data, or common block names
(2) External link components
(3) Link names
(4) Filenames

An alphabetical list of reserved PRTE symbols follows:

ABORT
ACTCM
ADDCOM
ARGTP
BLANKS
BUTDIS
BUTHIT
BUTINT
CARG
CBUTHT
CHARS
CLERR
CLOCHK
CLOCK
CLPRM
CMFIND
CODEIN
COMINT
COMTAB
COMVAL
CVICH
DACTCM
DARG
DISER
DISERR
DISINT
DCTINT
DOTMV2
DSCHED
DSKED
DSKEDU
ERRCON
GRID
GRIDAT
ERP
ERRMES
FNAME
GETCH
GETDI
GETDP
GETFIL
GETLOG
GETSI
GETSR
GRID

IBITS
INBITS
JPSTAR
JUMPTO
HITBUT
KCLOSE
KEYIN
KEYST
KINIT
KREAD
LAYDAT
LAYOUT
LCLOSE
LINIT
LK1
LK2
LK3
LK4
LK5
LPHIT
LREAD
LTA.
MDLBUF
MDLVAR
MESDIS
MESINT
MNUDAT
MSTR
MVC
NMNU
NXTCHR
OPMNB
OUTMOD
PAGMNU
PARG
PBHIT
PFA.
PIGDNT
PIGDSP
PIGFIL
PIGS
PINT
PNUMB
POLL

PRCOMT
PRIN1
PROMP
PROMPI
PROMPT
PSHBUT
PSTRNG
PUSHB
PUTCH
PUTDI
PUTDP
PUTSI
PUTSR
QUIT
RDMNU
RESLIB
RESOLV
SCHARR
SCHED
SKEDL
SKEDUL
STYLI
STYLS
SYSDNT
TARGS
TCLOSE
TERMIN
TIMEB
TIMEP
TINIT
TREAD
USEGRI
USEKEY
USESTY
VTERR
WDAT
WINK
WMNU
WRTUM
WTMNU
XCROSS
YYCLIN
YYBCD
YYFLA
YYPRLN
YYSTRT

APPENDIX I MENDEL COMMAND LIST

COMMAND NAME	ARGS	CONTEXT	VALID CREATE	VALID EDIT	FUNCTION	MENDEL SUBROUTINE
CREAT	ANAME, NMENUS, NSUBBRS, NGLOBLS	+ 1			SETS MODE AND MNB FILE SIZE	YYCRE
EDIT	ANAME	+ 1			SETS MODE AND RENAMES APPLICATION	YYEDI
BIGBT		1	X		USE LARGE LIGHTBUTTONS, ABBREVIATE MODE	YYBIG
ABREV	ALOGIC*	1	X	X	ABBREVIATE COMMAND NAMES TO 5 CHARACTERS	YYABR
KEYB	AKEYBD*,AECHO*	1	X	X	DEFINE ACTIVE KEYBOARD DEVICE AND ECHO ON TTA	YYKEY
STYLS	ASTYLS*	1	X	X	DEFINE ACTIVE STYLUS DEVICE	YYSTY
DELAY	NMILSEC	1	X	X	SET DELAY IN MILLISECONDS AFTER COMMAND SELECTION	YYDLA
SAVE	ALOGIC*	1	X	X	WRITE MENU TO DISC ON MENU CHANGE	YYSAVE
MNDEC	AMENU, AMENU...	+ 2	X	X	DECLARE MENU NAMES	YYMND
SBDEC	ASUBBR, ASUBBR...	+ 3	X	X	DECLARE APPLICATION SUBROUTINE NAMES	YYSBD
GLOBL		+ 4	X	X	BEGINS DEFINITION OF GLOBAL COMMANDS	YYGLO
ARGET		+ 5	X	X	NOT IMPLEMENTED, BUT MUST BE PRESENT IN CREATE MODE	YYARG
MENU	AMENU	+ 6	X	X	BEGINS DEFINITION OF NAMED MENU	YYMEN
DSABL	ADEV*, ADEV*	6	X	X	DEFINES ACTIVE DEVICES FOR MENU. NAMED ARE DISABLED	YYDSAB
SVMNU	ALOGIC*	≥ 5	X	X	CURRENT MENU TO BE WRITTEN TO DISC ON MENU CHANGE	YYSVMN
ENTER		≥ 5	X	X	POSITIONS COMMAND CURSOR AT ENTRY COMMAND	YYENT
EXIT		≥ 5	X	X	POSITIONS COMMAND CURSOR AT EXIT COMMAND	YYEXIT
PUSHB	NBUTTON	≥ 5	X	X	POSITIONS COMMAND CURSOR AT PUSHBUTTON NUMBER	YYPUSH
LOCAL	NLOCAL	≥ 5	X	X	POSITIONS COMMAND CURSOR AT LOCAL NUMBER	YYLOCL
COM	ANAME, NARGS, ADOCODE*, ASUBBR, AMNUCODE*, AMENU, ADATA	≥ 4	X	X	DEFINES A COMMAND AT CURRENT CURSOR POSITION AND STEPS TO NEXT COMMAND	YYCOM
END	AMENU	+ Ø	X	X	TERMINATES MENU DEFINITION. DEFINES STARTING MENU	YYEND
POS	NRELATIVE OR ACOMNAME	≥ 4		X	DISPLACES THE COMMAND CURSOR RELATIVE TO CURRENT POSITIONS CURSOR AT COMMAND NAME	YYPOS YYPOS
TOP		≥ 4		X	POSITIONS COMMAND CURSOR AT FIRST GLOBAL OR MENU COMMAND	YYTOP
BOT		≥ 4		X	POSITIONS COMMAND CURSOR AT LAST GLOBAL OR MENU COMMAND	YYBOT
REP	ANAME, NARGS, ADOCODE*, ASUBBR AMNUCODE*, AMENU, ADATA	≥ 4		X	REPLACES COMMAND AT CURRENT POSITION CURSOR UNCHANGED	YYREP
DEL		≥ 4		X	DELETES COMMAND AT CURRENT POSITION. CURSOR UNCHANGED	YYDEL
FIN		ANY		X	TERMINATES AN EDIT	

NOTES

1. Arguments

(a) Underlined arguments *must* be specified.
All other arguments assume default values given in Chapter 2.

(b) Arguments beginning with the letter 'A' are strings, with the letter 'N' are numbers.

(c) Arguments succeeded by ... may be repeated indefinitely.

(d) Square brackets enclosing arguments means a choice.

(e) Arguments succeeded by an asterisk must be one of a set of special strings:

Argument	Permissible strings
ALOGIC	TRUE FALSE
AKEYBD	TTA LTA
AECHO	ECHO ONLY
ASTYLS	VWA LPN
ADEV	KEYB LTBUT PUSHB CLOCK
ADOCODE	DO DONOW DONT DTNOW
AMNUCODE	MENU ENTER EXIT GO

2. Context

Context numbers preceded by '+' indicate that the command terminates the numerically preceding context and begins the context specified. +Ø terminates context 6 and enters no context.

APPENDIX J - MENDEL SUBROUTINES

These routines are contained in UPDATE file CMENDL BIN ON DECtape 16Ø

NAME	DESCRIPTION
CHARS	Block data for common block CHARS
DFHEAD	Sets up default MNB header block values
ERRMES	Issues error message numbers. Contains entry point ABORT. Filename is ERRMS3
GETFIL	Retrieves filename from argument common. Same as in .LIBRP
GETLOG	Retrieves filename
ISUP	Computes the least integer greater than
MDLRAM	Jump Table for YYMDLA. Contains addresses of all MENDEL application procedures
MENDEL	Editor-assembler main program. Parses option string and executes options
PAGMNU	Block data for common block PAGMNU
PROMPT	Dummy PROMPT routine for MENDEL. Filename is PRPDUM
STRAD	Converts a 5/7 ASCII string into radix 5Ø loader format
YYABR	Interprets ABREV command
YYADCD	Adds a loader code and data item to the relocateable binary output buffer
YYARG	Interprets the ARGET command
YYBIG	Interprets the BIGBT command
YYBOT	Interprets the BOT command
YYCARG	Compares a 5/7 ASCII string against a list of strings
YYCERR	Issues an error on illegal context
YYCL	Fills the Command Table with null commands
YYCLOS	Closes all disc files
YYCOM	Interprets the COM command
YYCOMS	Produces a loader code to declare a common block size
YYCONS	Produces a loader code to define a constant
YYCRE	Interprets the CREAT command

Cont'd

APPENDIX J - MENDEL SUBROUTINES

NAME	DESCRIPTION
YYDEL	Interprets the DEL command
YYDLA	Interprets the DELAY command
YYDPMN	Writes the 5/7 ASCII Dump File to disc
YYDSAB	Interprets the DSABL command
YYEDI	Interprets the EDIT command
YYEND	Interprets the END command
YYENT	Interprets the ENTER command
YYEPG	Produces a loader code to end program definition and flushes the output buffer
YYEXIT	Interprets the EXIT command
YYFLSH	Writes the relocateable binary output buffer to disc and clears the buffer
YYGLO	Interprets the GLOBL command
YYGTB	Maps command cursor index into global menu block needed and inputs the block
YYHOUT	Writes out MNB file header block if in EDIT mode
YYIGLB	Produces a loader code to declare an internally defined global
YYIN	Inputs a 25Ø word disc block
YYINTP	Initialises the relocateable binary output buffer
YYISYM	Produces a loader code to declare an internal symbol
YYJMPI	Produces a relocateable binary Jump Table from the SNT and writes it to slot 13
YYKEY	Interprets the KEYB command
YYLCB	Searches menu blocks for a global or local command name
YYLCM	Searches a single menu block for a command name
YYLMN	Searches the MAT for a menu name and returns its block number
YYLOCL	Interprets the LOCAL command
YYLOG	Interprets command LOG (not implemented)
YYLSB	Searches the SNT for an application procedure name and returns its index

Cont'd

APPENDIX J - MENDEL SUBROUTINES

NAME	DESCRIPTION
YYMDLA	Reads, parses, and interprets MENDEL program and produces an MNB file
YYMEN	Interprets the MENU command
YYMND	Interprets the MNDEC command
YYMNUL	Outputs a listing of the MAT on slot 13
YYNCH	Determines the number of non-blank characters in a 5/7 ASCII symbol
YYNCX	Issues error message 19 if wrong MENDEL context
YYNED	Issues an error message if not in edit mode
YYOPEN	Opens an MNB file and reads the header block into core
YYOUT	Writes a menu block to disc
YYPLOD	Produces a loader code to set the program load address
YYPNAM	Produces a loader code to name a program
YYPOS	Interprets the POS command
YYPRSZ	Produces a loader code to define program core size
YYPUSH	Interprets the PUSHB command
YYRELI	Produces a loader code to define a relocateable instruction
YYREP	Interprets the REP command
YYRVEC	Produces a loader code to define a relocateable transfer vector
YYSAVE	Interprets the SAVE command
YYSBD	Interprets the SBDEC command
YYSBRL	Outputs a listing of the SNT on slot 13
YYSHC	Outputs the decoded command node at the command cursor position
YYSPC	Creates a free command node by moving other nodes in a menu block
YYSRCH	Searches a 5/7 ASCII string for a particular character and returns its index
YYSTY	Interprets the STYLS command
YYSVMN	Interprets the SVMNU command
YYSYMB	Produces a loader code to declare a 5/7 ASCII string as a radix 50 symbol
YYTOP	Interprets the TOP command
YYWTB	Writes a binary 250-word record to disc on slot 13
YYXGLB	Produces a loader code to reference an externally defined global symbol
YYYABO	Interprets the ABORT command (not used)
YYYINT	Dummy initialisation routine (not used)

APPENDIX K - PIGS ERROR MESSAGES

ERROR NUMBER	DESCRIPTION	ACTION TAKEN	PRTE OR MENDEL SOURCE ROUTINE
1	Keyboard input buffer overflow. Too many characters typed	Command ignored	KEYIN
2	Format error in command string	Command ignored	TARGS
3	Index of argument to be retrieved is .LT. Ø or .GT. 14	Default argument returned	GETDP,GETCH
4	Number argument to be retrieved, but string argument input	Default argument returned	GETDP
5	Number argument to be retrieved would overflow integer variable	Default argument returned	GETSI
6	Command index in Command Table out of range	Command ignored	PIGS
7	Menu block number out of range	New menu not actuated	PIGS
8	Command name not recognized	Command ignored	PIGS
9	String argument to be retrieved, but number argument input	Command ignored	GETCH
10	Index of argument to be put is .LT. Ø or .GT. 14	Argument is type omitted	PUTDP,PUTCH
11	A necessary argument to this command was omitted	Command ignored	YYCRE,YYDSAB YYEDI,YYMEN YYMND,YYSBD
12	String argument would overflow buffer if retrieved	Default argument returned	GETCH
13	All menu blocks allocated are full	Command ignored	YYOUT
14	Reference to an undeclared application procedure name	Command ignored	YYCOM
15	Reference to an undeclared menu name	Command ignored	YYCOM,YYEND, YYMEN

Cont'd

APPENDIX K - PIGS ERROR MESSAGES

ERROR NUMBER	DESCRIPTION	ACTION TAKEN	PRTE OR MENDEL SOURCE
16	Unrecognized or illegal argument for this command	Command ignored	YYCOM,YYDSAB, YYKEY,YYSTY
17	Argument value out of range	Command ignored	YYLOCL,YYPUSH
18	Menu block full, command not entered	Command ignored	YYCOM
19	Illegal command for this MENDEL context	Command ignored	YYCERR,YYNCX
20	Command name not found, command cursor unchanged	Command ignored	YYPOS
21	Would position cursor off last menu block. Cursor unchanged	Command ignored	YYGTB,YYPOS
22	Command illegal in MENDEL edit mode	Command ignored	YYBIG
23	Command illegal in MENDEL create mode	Command ignored	YYNED
24	End of block reached by DEL command, cursor unchanged	Command ignored	YYDEL
25	File not found	MENDEL - Retype option string PRTE- Ignore read request	YYMDLA,OPMNB
26	File read error	MENDEL - Abort assembly or edit PRTE - Ignore read request	YYIN,YYMDLA RDMNU
27	Unexpected EOF on input file	MENDEL - Abort assembly or edit PRTE - Ignore read request	YYIN,RDMNU

Cont'd

APPENDIX K - PIGS ERROR MESSAGES

ERROR NUMBER	DESCRIPTION	ACTION TAKEN	PRTE OR MENDEL SOURCE
28	File write error	MENDEL - Abort assembly PRTE - Ignore write request	YYOUT,WTMNU
29	MNB file size error	Ignore file request	OPMNB
30	Clock schedule full or time interval too large	Command not scheduled	SCHEDL

APPENDIX K - PIGS ERROR MESSAGES

Error Numbers between 101 and 199 are FOG Errors

ERROR NUMBER	DESCRIPTION	ACTION TAKEN	FOG SUBROUTINE SOURCE
101	Wrong number of arguments	Request ignored	ALL
102	Badly formed file: DJMP* order not where it should be. Possible bad file length	Request ignored	ALL
103	Length of file (first array element value) .LT. Ø	Display file undefined	DCHOOS
104	Length of text string .LT. Ø	No code generated	TEXT,ITEXT
105	Illegal save-restore code	Save and restore	DRAW,RDRAW,IDRAW
106	Array index value out of range	Display file undefined	DCHOOS
107	Instructions to be inserted would overflow 8K bank : boundary or array dimension	No code generated	CODE PRODUCING ROUTINES
110	Illegal display class code	No code replaced	'I' prefixed routines
111	No current display file address, probably because of missing or faulty DCHOOS	No code generated	Code producing routines
112	FOG save register number out of range	Request ignored	SCHOOS,SINIT RCHOOS,RINIT
113	FOG save register referenced is undefined, probably because of missing or faulty SCHOOS or SINIT	Request ignored	RCHOOS,RINIT

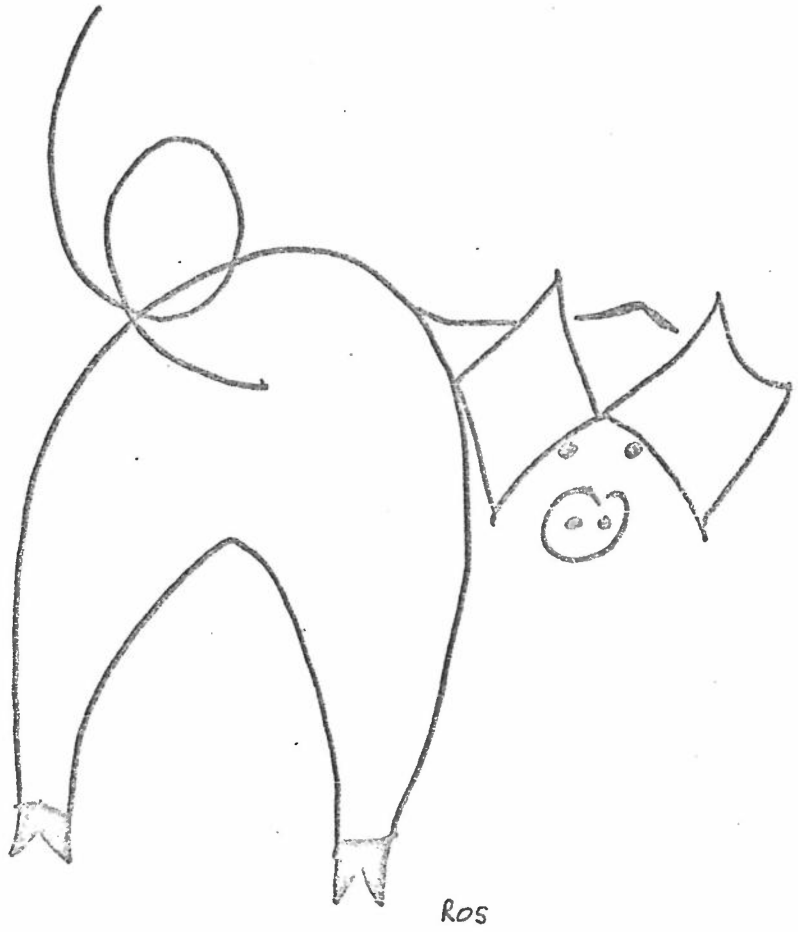
Error numbers greater than 199 are GAP errors.
IOPS and OTS ERRORS are explained in the DOS and FORTRAN manuals, respectively

ACKNOWLEDGEMENT

The author wishes to thank F R A Hopgood and J R Gallop for their valuable advice in the design and programming of PIGS.

The manual was prepared with great patience by:

J B Chamberlain	Organisation
G M Miles	Typing
K M Gascoigne	Photocopying



THE END