

Professional™
300series

**Professional Developer's
Tool Kit User's Guide**

Order No. AA-N617D-TK

Developer's Tool Kit

digital
software

Professional Developer's Tool Kit User's Guide

Order No. AA-N617D-TK

April 1984

This guide explains how to use the Professional Developer's Host Tool Kit and the PRO/Tool Kit to write application software that will run under the Professional Operating System (P/OS).

DEVELOPMENT SYSTEM: VAX/VMS V3.0 or later
RSX-11M V4.0 or later
RSX-11M-PLUS V2.0 or later
P/OS V2.0

SOFTWARE VERSION: Host Tool Kit V2.0
PRO/Tool Kit V2.0

First Printing, December 1982
Revised, May 1983
Revised, September 1983
Revised, April 1984

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software or equipment that is not supplied by DIGITAL or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for manufacture or sale of items without written permission.

Copyright © 1982, 1983, 1984 by Digital Equipment Corporation
All Rights Reserved

The following are trademarks of Digital Equipment Corporation:

CTIBUS	MASSBUS	Rainbow
DEC	PDP	RSTS
DECmate	P/OS	RSX
DECsystem-10	PRO/BASIC	Tool Kit
DECSYSTEM-20	PRO/Communications	UNIBUS
DECUS	Professional	VAX
DECwriter	PRO/FMS	VMS
DIBOL	PRO/RMS	VT
digital	PROSE	Work Processor
	PROSE PLUS	

CONTENTS

PREFACE	viii
-------------------	------

CHAPTER 1 GETTING STARTED

1.1	OVERVIEW	1-1
1.2	TOOL KIT CONFIGURATIONS	1-1
1.3	THE TARGET SYSTEM	1-2
1.4	THE DEVELOPMENT CYCLE	1-3
1.5	USING THE PRO/TOOL KIT	1-4
1.6	USING THE HOST TOOL KIT	1-4
1.7	THE TOOL KIT DOCUMENTATION SET	1-5
1.7.1	Volume 1: Introduction	1-5
1.7.2	Volume 2: Video	1-7
1.7.3	Volume 3: Task Building	1-7
1.7.4	Volume 4: P/OS System	1-8
1.7.5	Volume 5: PRO/RMS-11	1-8
1.7.6	Volume 6: MACRO Program Development	1-9
1.7.7	Volume 7: PRO/DECnet	1-10
1.7.8	PRO/Tool Kit Volume	1-11

CHAPTER 2 THE PROFESSIONAL DEVELOPER'S TOOL KIT

2.1	LANGUAGES	2-1
2.1.1	BASIC-PLUS-2	2-1
2.1.2	COBOL-81	2-1
2.1.3	DIBOL	2-2
2.1.4	FORTRAN-77	2-2
2.1.5	MACRO-11	2-2
2.1.6	PASCAL	2-3
2.2	SOFTWARE DEVELOPMENT TOOLS	2-3
2.2.1	Application Builder (PAB)	2-3
2.2.2	Application Diskette Builder	2-3
2.2.3	Communications	2-3
2.2.4	CORE Graphics Library (CGL)	2-4
2.2.5	DECnet	2-5
2.2.6	General Image Display Instruction Set (GIDIS)	2-5
2.2.7	Fast Install	2-6
2.2.8	File Control Services (FCS-11)	2-6
2.2.9	Forms Management System (PRO/FMS-11)	2-6
2.2.10	Frame Development Tool (FDT)	2-7
2.2.11	On-Line Debugging Tool (ODT)	2-7
2.2.12	Print Services Callable Task	2-7
2.2.13	POSRES User Interface Services Library	2-7
2.2.14	POSSUM System Services Library	2-8
2.2.15	PROSE Callable Editor Task	2-8
2.2.16	Record Management Services (PRO/RMS-11)	2-8
2.2.17	SORT Callable Sort Task	2-8

CHAPTER 3	THE PROFESSIONAL 300 SERIES PERSONAL COMPUTER	
3.1	HARDWARE CONFIGURATIONS	3-1
3.2	THE PROFESSIONAL OPERATING SYSTEM (P/OS)	3-1
3.2.1	The P/OS User Interface	3-3
3.2.2	P/OS Services	3-3
3.2.3	P/OS System Components	3-4
3.2.4	Calling P/OS Routines from High-level Languages	3-7
3.2.5	Calling P/OS Routines from MACRO-11	3-10
 CHAPTER 4	 THE APPLICATION DEVELOPMENT CYCLE	
4.1	THE DESIGN PHASE	4-1
4.2	THE IMPLEMENTATION PHASE	4-1
4.3	THE BUILD PHASE	4-2
4.4	THE TESTING PHASE	4-3
4.4.1	Testing Applications on the PRO/Tool Kit	4-3
4.4.2	Testing Applications on the Host Tool Kit	4-4
4.5	THE DEBUGGING PHASE	4-5
4.6	THE TUNING PHASE	4-5
4.7	THE DISTRIBUTION PHASE	4-5
 CHAPTER 5	 APPLICATION DESIGN CONSIDERATIONS	
5.1	TARGET SYSTEM CONFIGURATIONS	5-1
5.2	VIRTUAL ADDRESS SPACE	5-2
5.2.1	Overlaying	5-2
5.2.2	Cluster Libraries	5-3
5.2.3	Multiple Tasks	5-3
5.2.4	Multiple Regions	5-3
5.3	PHYSICAL MEMORY	5-4
5.3.1	Checkpointing	5-4
 CHAPTER 6	 USING APPLICATION FILES	
6.1	P/OS FILE SPECIFICATIONS	6-1
6.1.1	Devices	6-2
6.1.2	Directories	6-4
6.1.3	File Names	6-5
6.1.4	File Types	6-6
6.1.5	Version Numbers	6-6
6.1.6	Wild Card Conventions	6-7
6.2	LOGICAL NAMES	6-7
6.2.1	System Logical Names	6-8
6.2.2	PRO/RMS-11 Translation of Logical Names	6-9
6.2.3	FILES-11 ACP Use of Logical Names	6-10
6.2.4	Manipulating Logical Names	6-10
6.3	ACCESSING APPLICATION FILES	6-11
6.3.1	Menu and Help Files	6-11

6.3.2	Running Applications from the PRO/Tool Kit . . .	6-12
-------	--	------

CHAPTER 7 TASK BUILDING

7.1	INVOKING PAB ON THE PRO/TOOL KIT	7-1
7.2	INVOKING PAB ON VAX/VMS	7-1
7.3	INVOKING PAB ON RSX-11M/M-PLUS (DCL)	7-2
7.4	BUILDING APPLICATIONS	7-2
7.5	THE COMMAND (.CMD) FILE	7-3
7.5.1	The Command Line	7-3
7.5.2	The CLSTR Option	7-4
7.5.3	NULLIB	7-5
7.6	THE OVERLAY DESCRIPTOR LANGUAGE FILE	7-6

CHAPTER 8 P/OS USER INTERFACE SERVICES

8.1	OVERVIEW	8-1
8.2	DESIGNING A MENU STRUCTURE	8-4
8.2.1	Format of a Menu	8-4
8.2.2	Single-Choice Menus	8-5
8.2.3	Multiple-Choice Menus	8-6
8.2.4	Key Processing in Menus	8-7
8.3	IMPLEMENTING A MENU STRUCTURE	8-9
8.3.1	Displaying Menus	8-9
8.3.2	Programming with Menus	8-11
8.3.3	File Specification Routines	8-12
8.4	DESIGNING A HELP STRUCTURE	8-14
8.4.1	Help Menus	8-14
8.4.2	Key Processing in Help Menus	8-15
8.4.3	Help Text Frames	8-16
8.4.4	Key Processing in Help Text Frames	8-16
8.4.5	A Sample Help Structure	8-17
8.5	IMPLEMENTING A HELP STRUCTURE	8-19
8.5.1	Opening Help Files	8-19
8.5.2	Setting the Default Help Frame	8-19
8.5.3	Activating the Help Structure	8-20
8.6	MESSAGE FILES AND SERVICES	8-21
8.7	FUNCTION KEYS	8-22
8.7.1	Using Function Keys	8-23
8.7.2	Programming Function Keys	8-24
8.8	POSRES TASK IMAGE REQUIREMENTS	8-25
8.8.1	The UNITS Option	8-26
8.8.2	The GBLDEF Option	8-26
8.8.3	The ASG Option	8-27
8.8.4	The EXTSCAT Option	8-28
8.8.5	Placing Buffers in Overlay Branches	8-31

CHAPTER 9

APPLICATION TUNING

9.1	SYSTEM DIRECTIVES	9-1
9.1.1	Cost of Directives	9-1
9.1.2	Directives .vs. Servers	9-2
9.2	FILE HANDLING	9-2
9.2.1	When to Open Files	9-3
9.2.2	Use of File ID's	9-3
9.2.3	File Pre-Allocation	9-3
9.2.4	Pre-extending	9-3
9.2.5	Multiblock I/O	9-4
9.3	VIDEO PERFORMANCE	9-4
9.3.1	Size of Buffer	9-4
9.3.2	Buffering	9-5
9.3.3	Eight Bit Escape Sequence Characters	9-6
9.3.4	QIO\$.vs. QIOW\$	9-7
9.3.5	Turning the Cursor Off	9-7
9.3.6	Standard Video Tricks	9-8
9.4	KEYBOARD INPUT	9-9
9.4.1	Function Keys	9-9
9.4.2	Eight Bit Characters	9-9
9.4.3	Input Buffering	9-10
9.4.4	AST's With Notification	9-10
9.5	PROGRAMMABLE LOGICAL ADDRESS SPACE (PLAS) CONSIDERATIONS	9-10
9.6	MULTI-TASK APPLICATIONS	9-11
9.6.1	Intertask Communication	9-11
9.6.2	Memory usage	9-12
9.6.3	Significant event impact	9-12
9.6.4	Contention	9-13
9.7	CONCURRENT APPLICATIONS	9-14
9.7.1	Only for "bounded" systems	9-14
9.7.2	Exercise extreme care	9-14
9.8	POOL CONSIDERATIONS	9-15
9.8.1	Offspring Control Blocks	9-15
9.8.2	Lock Blocks	9-15
9.8.3	Open Files	9-16
9.8.4	Attachment Descriptor Blocks	9-16
9.8.5	Send Data Packets	9-16
9.8.6	I/O Packets	9-16

APPENDIX A

GLOSSARY

APPENDIX B

SPACE REQUIREMENTS FOR P/OS DISKETTE V1.7

APPENDIX C

POSRES STATUS BLOCK CODES

APPENDIX D FUNCTION KEY NAMES AND CODES

APPENDIX E DOCUMENTATION DIRECTORY

E.1	VOLUME 1: INTRODUCTION	E-1
E.2	VOLUME 2: VIDEO	E-1
E.3	VOLUME 3: TASK BUILDING	E-1
E.4	VOLUME 4: P/OS SYSTEM	E-2
E.5	VOLUME 5: PRO/RMS-11	E-2
E.6	VOLUME 6: MACRO PROGRAM DEVELOPMENT	E-2
E.7	VOLUME 7: PRO/DECNET	E-2
E.8	PRO/TOOL KIT VOLUME	E-2

INDEX

FIGURES

1-1	Tool Kit Configurations	1-2
3-1	Professional Keyboard (U.S./Canada)	3-2
7-1	Sample PAB Command File	7-3
8-1	The User Interface Tools	8-3
8-2	Single-choice Menu	8-5
8-3	Multiple-choice Menu	8-6
8-4	"Name a File Form"	8-13
8-5	Help Menu	8-15
8-6	Help Text Frame	8-17
8-7	The P/OS Main Menu Help Structure (partial)	8-18
8-8	Message Frame	8-21
8-9	PAB Command File with POSRES Options	8-26
8-10	POSRES Maximum Buffer Sizes	8-29
8-11	Sample .ODL File Showing Overlaid Buffers	8-32

TABLES

6-1	P/OS Physical and Pseudo Device Names	6-3
6-2	User-visible File Types	6-6
6-3	System/Application File Types	6-6
6-4	P/OS System Logical Device Names	6-8
6-5	Logical and Equivalence Directory Names	6-9
8-1	POSRES Global Symbols	8-27
8-2	Buffers Accessed by POSRES Routines	8-29
C-1	POSRES Status Values	C-1
C-2	Menu Service Routine Errors	C-3

PREFACE

Intended Audience

The Tool Kit User's Guide is for application developers who are creating software that will run under P/OS. It assumes that you are familiar with the host system environment. If you need introductory system information, refer to the RSX-11M/M-PLUS Guide to Program Development, your host system command language manual, or the PRO/Tool Kit Command Language and Utilities Manual.

Document Structure

The Tool Kit User's Guide contains the following chapters and appendices:

1. Getting Started explains what the Tool Kit is all about. It provides an overview of the Tool Kit configurations, the target system, the development cycle, describes how to invoke the Tool Kit, and lists the contents of the documentation set.
2. The Professional Developer's Tool Kit provides a brief overview of each programming language and software development tool that makes up the Tool Kit.
3. The Professional 300 Series Personal Computer is the target system for the Tool Kit. This chapter describes the hardware and software that makes up a Professional.
4. The Application Development Cycle is a series of phases that you will go through in developing an application for the Professional. This chapter provides an overview of each phase.
5. The first phase of application development is design. This chapter covers some of the factors that you should consider during the design phase.
6. Using Application Files explains P/OS file specification syntax, logical names, and how an executing application opens files.
7. Task Building provides a brief explanation of how to use the Application Builder, the utility that creates task images.
8. P/OS User Interface Services describes how to design and implement a menu-based user interface for your application, similar to that used by P/OS itself.
9. The penultimate phase of application development is tuning. This chapter explains some of the factors that affects an application's performance. It is intended for experienced programmers.

PREFACE

- A. The Glossary describes some of the terms used in this manual
- B. The Space Requirements for P/OS Diskette are significant if you want your application to run on diskette-based systems.
- C. The POSRES Status Block Codes describes the status information returned by the User Interface Library Routines at run-time. (This appendix is duplicated in the Tool Kit Reference Manual).
- D. Function Key Names and Codes lists the Professional function keys in a format useful for user interface programming. (This appendix is duplicated in the Tool Kit Reference Manual).
- E. The Documentation Directory lists the volumes, manuals, updates, and order numbers of the Tool Kit documentation set.

Related Documentation

This manual is part of the Tool Kit documentation set. Related documentation is referenced throughout the book. For abstracts and order numbers of other Tool Kit documents, see Appendix E.

Documentation Conventions

Syntax diagrams are presented in a format intended to make them easy to read and understand. Most languages do not require that you format your code in any particular way; therefore, you should not regard the formats used in this manual as mandatory.

Because some languages include the symbols normally used as conventions in syntax diagrams, the following conventions are used:

Convention	Meaning
BOLD UPPERCASE	Bold uppercase letters indicate elements that you must use exactly as shown.
UPPERCASE LETTERS	Uppercase letters indicate elements that you can omit or use exactly as shown.
bold lowercase	Bold lowercase letters indicate elements that you must replace according to the description in the text.
lowercase letters	Lowercase letters indicate elements that you can omit or replace according to the description in the text.

PREFACE

•
•
•

A vertical ellipsis in a figure or example means that not all of the statements are shown.

red letters

Red letters distinguish what you type from what the computer types.

CHAPTER 1

GETTING STARTED

1.1 OVERVIEW

The Professional Developer's Tool Kit is a set of programming languages and software tools for developing applications for the Professional 300 Series personal computers. The Professional consists of a PDP-11/23-PLUS processor running P/OS, an operating system based on RSX-11M-PLUS, a real-time, multitasking system. Thus, the Tool Kit consists of software products that were originally designed for the PDP-11 minicomputer family, including much larger systems like the PDP-11/70. Most of the tools are RSX-11M-PLUS layered products that have been modified for Professional 300 Series applications.

If you have had experience with PDP-11 minicomputers and RSX-11M-PLUS, you will find the Tool Kit and P/OS very similar to tools that you have already used. However, if your background has been primarily with other personal computers, you may find that the Tool Kit and P/OS are much more powerful and complex than what you have been using.

The Tool Kit provides an extensive documentation set that describes the tools and how to use them to develop applications (overview provided in Section 1.7). This manual and the Tool Kit Reference Manual make up the "core" of the documentation set. If you are new to the Tool Kit and P/OS, it is recommended that you become familiar with them first.

Some tools have documentation consisting of several manuals. In those cases, the RSX-11M/M-PLUS or generic manuals are included along with a supplement containing information specific to the Tool Kit version.

TOOL KIT CONFIGURATIONS

1.2 TOOL KIT CONFIGURATIONS

The Tool Kit is available in two configurations:

- The PRO/Tool Kit runs as an installed application on the Professional 350 personal computer and uses the DIGITAL Command Language (DCL) as its user interface. It allows you to use the Professional 350 as a development system and a target system at the same time.
- The Host Tool Kit runs on a VAX/VMS system (in PDP-11 compatibility mode) or on a PDP-11 system running RSX-11M/M-PLUS. You can take advantage of the performance, mass storage, and communications provided by a minicomputer development system while using the Professional 350 as a terminal and a target system at the same time.

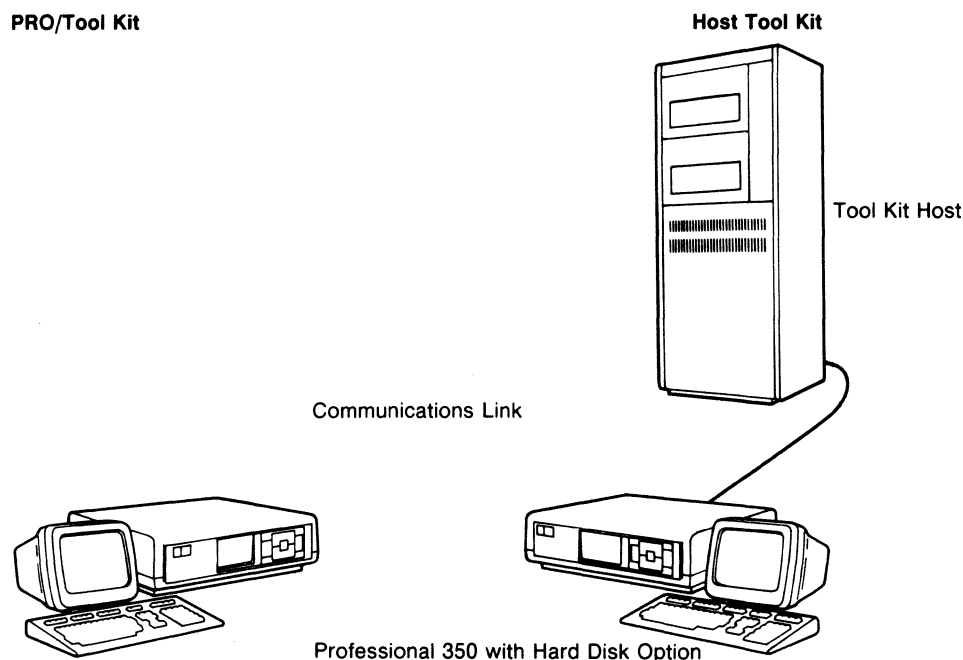


Figure 1-1: Tool Kit Configurations

1.3 THE TARGET SYSTEM

The Professional 300 Series of personal computers has two hardware configurations: the Professional 325 and the Professional 350. Each system unit comes with a dual diskette drive. The primary difference is the mass-storage device: the Professional 350 can support a hard disk.

THE TARGET SYSTEM

The Professional 300 Series also has two software configurations: P/OS Hard Disk and P/OS Diskette. You must take this into consideration when designing your application.

1.4 THE DEVELOPMENT CYCLE

The Tool Kit development cycle consists of several phases. Each phase is described in detail in Chapter 4.

- **The Design Phase**

The design phase requires you to make several decisions that will affect all of the other phases. They are: target system configurations and use of virtual and physical memory.

- **The Implementation Phase**

During the implementation phase, you create a set of files that specifies the algorithms and data structures used in your application: source code, command files, form description files, frame definition files, data files, and so forth. Some of these files will change continuously as you work. Others will remain relatively stable.

- **The Build Phase**

During the build phase, you create another set of files that represents an actual working version of your application: executable images, form libraries, converted frame files, and so forth.

- **The Test Phase**

During the test phase, you install the working version of your application on a Professional personal computer and test all aspects of it.

- **The Debugging Phase**

Some of the Tool Kit languages offer special debugging software that allows you to stop and start an executing task, examine variables, and so forth. You can connect a separate terminal to your Professional for debugger interaction, allowing your task to control the video monitor.

THE DEVELOPMENT CYCLE

- **The Tuning Phase**

As your application nears completion, you may want to make some adjustments to optimize its performance and use of resources. If so, make sure to retest it thoroughly.

- **The Distribution Phase**

When you are satisfied that your application is ready to ship, create a master distribution kit for duplication. The Tool Kit includes a special tool for this purpose.

1.5 USING THE PRO/TOOL KIT

All the tools you need for program development, including the DIGITAL editor EDT, are included in the PRO/Tool Kit. You can complete the entire application development cycle without leaving the PRO/Tool Kit environment.

Start up your system and select the PRO/Tool Kit option from the appropriate menu. The startup command procedure START.CMD prints messages indicating which tasks are being installed. The dollar sign (\$) prompt indicates that the Digital Command Language (DCL) user interface is ready to accept commands. For complete information on DCL, refer to the PRO/Tool Kit Command Language/Utilities Manual.

If you would like to have the PRO/Tool Kit (or any other application) run automatically when P/OS starts up, create a file on the hard disk named:

```
[ZZSYS]FIRSTAPPL.PTR
```

In the file, insert one line of text, exactly nine characters long (padded with spaces if necessary) that specifies the name of the directory (no brackets) containing the application that you want to run automatically. For example, start the PRO/Tool Kit and type:

```
$ SHOW LOGICAL APPL$DIR
```

If the PRO/Tool Kit directory is [ZZAP00002], the file should contain:

```
ZZAP00002
```

NOTE

Be careful when creating this file. If you make a mistake, P/OS usually displays the Main Menu. It may be possible, however, to render a system unusable if the startup application creates problems.

USING THE HOST TOOL KIT

1.6 USING THE HOST TOOL KIT

The Host Tool Kit requires that you work on a host computer system for the build phase of the development cycle. Most developers find it convenient to use the host for all phases, right up to testing/debugging.

For host system work, you can use whatever terminal you prefer. It is recommended that you use a Professional 350 personal computer and Terminal Emulator for your host work. This allows you to work interactively with eight-bit characters. If you use a VT100-type terminal, you can work with eight-bit characters in files only.

Both VAX/VMS and RSX-11M/M-PLUS require that you enter some terminal commands to enable transmission or reception of eight-bit characters. See the PRO/Communications Manual for details on terminal commands.

1.7 THE TOOL KIT DOCUMENTATION SET

The Tool Kit documentation set is organized so that related manuals are in the same volume (binder). This section describes the contents of each volume.

You may want to refer to some manuals not included in the PRO/Tool Kit documentation set while you develop applications:

- **Professional 300 Series End User Documentation**

Read this documentation to learn how to install and run applications such as the PRO/Tool Kit on the Professional, and to understand the user's view of the system on which your applications will run.

- **Host System Documentation**

Complete documentation for RSX-11M/M-PLUS and VAX/VMS host systems can be obtained under separate license. The host Tool Kit documentation assumes that you have and are familiar with the manuals for your host system.

- **High-level Language Documentation**

Four optional high-level languages, Tool Kit COBOL-81, Tool Kit DIBOL, Tool Kit FORTRAN-77, and Tool Kit PASCAL are supported by the Professional Tool Kit. See your Tool Kit Software Product Description (SPD) for information on optional software products and documentation.

THE TOOL KIT DOCUMENTATION SET

1.7.1 Volume 1: Introduction

- **Host Tool Kit Installation Guide and Release Notes**

This manual contains instructions for installing the Tool Kit software on a host development system. It also provides information specific to the current release of the Professional Host Tool Kit. Ignore this manual if you are using the PRO/Tool Kit.

- **Tool Kit User's Guide**

This manual describes how to develop application software for the Professional 300 Series of personal computers. It contains chapters that describe how to get started with the Tool Kit, the individual tools, the target system, the development cycle, building tasks, installing applications, creating distribution kits, using application files, and the P/OS User Interface Services. This manual is intended for all Tool Kit users.

- **Tool Kit Reference Manual**

This manual provides reference information about the following tools: Application Diskette Builder, PRO/Communications, Fast Install, File Control Services (FCS), Frame Development Tool (FDT), Installation Command Languages, MACRO-11 Assembler (PMA), POSRES User Interface Library, Print Services, PROSE Text Editor, and PRO/SORT. This manual is intended for all Tool Kit users.

- **FMS-11/RSX Release Notes**

This manual provides information specific to the current release of the FMS-11/RSX software and the FMS-11/RSX User Environment Test Package (UETP).

NOTE

The Tool Kit installation procedure installs PRO/FMS-11. Do not perform the installation instructions in this manual.

- **PRO/FMS-11 Documentation Supplement**

This manual describes the differences between FMS/RSX and PRO/FMS, and should be used with the Tool Kit FMS-11/RSX documentation. It is intended for the application developer experienced with FMS-11.

THE TOOL KIT DOCUMENTATION SET

- **FMS-11/RSX Software Reference Manual**

This manual describes how to develop FMS-11 applications on RSX-11M and RSX-11M-PLUS systems. Use it along with the PRO/FMS-11 Documentation Supplement.

1.7.2 Volume 2: Video

- **Terminal Subsystem Manual**

This manual describes the hardware and software that controls the Professional keyboard and video monitor (the functions typically performed by a video terminal). It documents text mode, in which the Professional performs input/output using the DEC Multinational Character Set. It also describes the differences between VT102 and VT125 emulation on the Professional and the corresponding functions on VT102 and VT125 terminals.

- **CORE Graphics Library Manual**

This manual describes a general-purpose graphics subroutine library based on the ACM SIGGRAPH CORE Graphics Standard. The CORE Graphics Library provides a higher level graphics programming interface than PRO/GIDIS. It provides descriptions of fundamental graphics programming concepts and sample programs written in high-level languages.

- **PRO/GIDIS Manual**

This manual describes the General Image Display Instruction Set, a device-independent graphics interface that is specific to DIGITAL. PRO/GIDIS is intended for applications where speed and compactness are more important than the high-level implementation provided by the CORE Graphics Library. Instructions are encoded as a stream of op-codes and argument data, and passed from the application program to the PRO/GIDIS layer via the P/OS QIO mechanism. The manual is intended for developers with systems programming and graphics software experience. The sample programs are written in MACRO-11.

THE TOOL KIT DOCUMENTATION SET

1.7.3 Volume 3: Task Building

- **RSX-11M/M-PLUS Task Builder Manual**

This manual describes the RSX-11M/M-PLUS Task Builder (TKB) upon which the Professional Application Builder (PAB) is based.

NOTE

The Tool Kit User's Guide includes a chapter about task building. Use the RSX-11M/M-PLUS Task Builder Manual if you need more detailed information.

1.7.4 Volume 4: P/OS System

- **P/OS System Reference Manual**

This manual describes the P/OS Executive, the "nucleus" of the Professional Operating System. Topics include executive directives, system functions, checkpointing, memory allocation, the file system, and device drivers. It is intended for experienced application developers.

1.7.5 Volume 5: PRO/RMS-11

Only MACRO-11 assembly language programmers can use the full set of RMS-11 capabilities described in these manuals. All of the Tool Kit high-level languages provide an a built-in interface to RMS-11 that is appropriate for most applications.

NOTE

Some of the RSX-11 RMS-11 features documented in this manual may not be available to developers creating applications for P/OS. Refer to your Installation Guide and Release Notes for a list of any differences.

- **PRO/RMS-11: An Introduction**

This manual introduces the concepts of RMS-11 record formats, file organizations, and record access modes. It does not provide reference or usage information, but should be read before the other RMS-11 documents. It is intended for all users of RMS-11, including MACRO-11 and high-level language programmers.

THE TOOL KIT DOCUMENTATION SET

- **PRO/RMS-11 Macro Programmer's Guide**

This manual provides reference material about the macros and symbols that make up the interface between a MACRO-11 program and the RMS-11 operation routines. It is intended for application developers who are already familiar with RMS-11 facilities.

- **RSX-11M/M-PLUS RMS-11 Users's Guide**

This manual is a guide to using RMS-11 in file and task design for application programs written in either MACRO-11 or a high-level language. It is intended for application developers who are already familiar with RMS-11 facilities.

1.7.6 Volume 6: MACRO Program Development

- **IAS/RSX-11 ODT Reference Manual**

This manual describes how to use the On-line Debugging Tool (ODT) to debug user task images. It is intended for all application developers. Use it along with the IAS/RSX-11 ODT Supplement.

NOTE

Some high-level languages provide their own debugging facilities.

- **IAS/RSX-11 ODT Supplement**

This manual describes the differences between IAS/RSX-11 ODT and ODT on the Professional, and should be used with the IAS/RSX-11 ODT documentation.

- **PDP-11 MACRO-11 Language Reference Manual**

This manual describes how to use the MACRO-11 relocatable assembler to develop assembly language programs.

- **Guide to Writing a P/OS I/O Device Driver and Advanced Programmer's Notes**

This manual defines Executive and I/O driver interface protocols, describes system I/O data structures, and suggests I/O driver routine coding procedures. The information is provided in sufficient detail to allow you to:

THE TOOL KIT DOCUMENTATION SET

- Incorporate a user-written driver into a P/OS system.
- User Executive service routines that an I/O driver typically employs. employs.
- Develop applications that directly access the Professional video bitmap.

This manual is written for the senior-level system programmer who is familiar with the hardware characteristics of both the Professional 300 Series and the device that the user-written software supports. Unless explicitly noted otherwise, all information in this manual is subject to change without notice.

1.7.7 Volume 7: PRO/DECnet

DECnet is the name given to a family of software and hardware communications products that provide a network interface for Digital operating systems. DECnet enables multiple computer systems to participate in communications and resource sharing within a specific network. PRO/DECnet allows Professional 350 computers to connect to other DECnet systems on the Ethernet.

- **PRO/DECnet Tool Kit Release Notes**

This manual provides information specific to this release of the PRO/DECnet Tool Kit.

- **Introduction to DECnet**

This manual is an overview of the concepts and capabilities of DECnet networks. It describes the major network concepts behind all implementations of DECnet, defines specific network functions, and identifies the DECnet implementations that support each function.

- **PRO/DECnet Tool Kit Installation Guide**

This manual details procedures for installing the PRO/DECnet Tool Kit either on an RSX-11M/M-PLUS or a VAX/VMS host system, or on a Professional 350 personal computer. It also provides information on how to customize Professional systems as PRO/DECnet nodes.

THE TOOL KIT DOCUMENTATION SET

- **PRO/DECnet Programmer's Reference Manual**

This manual discusses software requirements for creating PRO/DECnet applications. It reviews software design conventions which are critical to the early stages of program development and details network programming calls used in the creation of PRO/DECnet applications. It assumes that you have a working knowledge of networking concepts.

1.7.8 PRO/Tool Kit Volume

This volume is provided only with the PRO/Tool Kit. It represents information that is otherwise available to users of the host Tool Kit through their host system documentation.

NOTE

Disregard this section if you are using the host Tool Kit.

- **PRO/Tool Kit Installation Guide and Release Notes**

This manual contains instructions for installing the PRO/Tool Kit software on a Professional 350 system. It also provides information specific to the current release of the PRO/Tool Kit.

- **PRO/Tool Kit Command Language and Utilities Manual**

This manual describes the Digital Command Language (DCL) and program development utilities available on the PRO/Tool Kit for all phases of application development.

- **RSX-11M/M-PLUS RMS-11 Utilities Manual**

This manual describes the RMS-11 Utilities available to users of RMS-11 on an RSX-11M/M-PLUS host system. The PRO/Tool Kit implementation is a subset of those utilities and is described in PRO/Tool Kit Command Language and Utilities Manual. This manual is intended for application developers who are using high-level languages and do not require or do not have access to the full set of RMS-11 capabilities.

CHAPTER 2

THE PROFESSIONAL DEVELOPER'S TOOL KIT

2.1 LANGUAGES

The following high-level languages must be ordered separately from the Tool Kit. MACRO-11 is part of the kit.

2.1.1 BASIC-PLUS-2

BASIC-PLUS-2 is an extended BASIC compiler. It takes full advantage of the floating point and integer instructions, as well as capabilities of the Professional Operating System.

In addition to elementary BASIC language features, BASIC-PLUS-2 provides compile-time directives, structured programming constructs, EXTERNAL statements, language subsets and subset flaggers, exception handling, and many more useful features for application development.

The BASIC-PLUS-2 language processor is composed of a compiler and an Object Time System Library. The compiler is included in the Professional Tool Kit BASIC-PLUS-2 product. The OTS is supplied as a cluster library with PRO/RMS-11 and P/OS.

2.1.2 COBOL-81

Professional Tool Kit COBOL-81 is a high-level language for business data processing. Compatible with COBOL-81/RSX, it also shares many features with VAX-11 COBOL; code developed using Tool Kit COBOL-81 may be migrated to VAX-11 COBOL.

COBOL-81 consists of a compiler and an Object Time System/Library. It uses an interactive symbolic debugger and includes many DIGITAL extensions to the COBOL language, including advanced screen handling, RMS-STX and RMS-STV special registers for debugging, and file sharing features to enable more than one task to access data at the same time.

LANGUAGES

2.1.3 DIBOL

Professional Tool Kit DIBOL is the Tool Kit version of DIGITAL's Business Oriented Language. Similar to COBOL in its use of DATA DIVISION and English-like procedural statements, DIBOL takes extensive advantage of the Professional's architecture.

DIBOL features RMS file support, a resident DIBOL library, and allows use of P/OS system services while maintaining many of the standard DIBOL features. It enables data manipulation, arithmetic expression evaluation, subroutines, structured constructs, table subscripting, record redefinition, external and internal calls to other programs, intertask communication, and random, sequential, and indexed access to files. In addition, DIBOL includes a comprehensive on-line debugging utility, DDT, with which you can quickly isolate and correct programming errors.

2.1.4 FORTRAN-77

Professional Tool Kit FORTRAN-77 is an extended implementation of the ANSI subset FORTRAN-77 standard. Switch selectable support is provided for user programs based on the previous ANSI FORTRAN standard.

Features include CHARACTER data types, Block IF constructs for the conditional execution of blocks of statements, double precision and complex data types, intrinsic functions, exponentiation forms, format edit descriptors, generic function selection, virtual array support and access to sequential, relative, and indexed organization files.

The compiler produces direct PDP-11 machine code optimized for execution-time efficiency on a Professional with a floating point adapter.

2.1.5 MACRO-11

Professional Tool Kit MACRO-11 (PMA) is the standard PDP-11 relocatable assembly language processor. The macro libraries, RSXMAC.SML and RMSMAC.MLB, reflect the changes in P/OS kernel system directives and in RMS-11 Version 2.0, respectively. PMA is supplied with the Tool Kit; you do not have to order it separately.

The assembler processes source statements sequentially, generating binary machine instructions and data words or performing assembly-time operations (such as macro expansions) for each statement.

LANGUAGES

2.1.6 PASCAL

Professional Tool Kit PASCAL is a structured, high-level language that provides a modular, systematic approach to problem solving. An extended implementation of PASCAL, it helps to simplify application design.

Major language features include user-defined and subrange scalar data types, structured variables and constants, loop control statements, standard functions and procedures, and attributes to facilitate access to P/OS resources. A full set of I/O routines support sequential, relative, and indexed files.

Tool Kit PASCAL is a true, optimizing compiler, generating PDP-11 instructions in the form of binary object modules or MACRO-11 source code. The object-time system is a clustered library.

2.2 SOFTWARE DEVELOPMENT TOOLS

2.2.1 Application Builder (PAB)

The Professional Application Builder (PAB) is the Tool Kit version of the RSX-11M/M-PLUS Task Builder, the utility that links user-created object modules with system software, producing task image files that can be executed on the Professional. PAB is an extremely powerful tool that provides sophisticated programmers with virtually unlimited control over the characteristics of a task image. It is documented briefly in this manual, and in detail in the RSX-11M/M-PLUS Task Builder Manual.

2.2.2 Application Diskette Builder

The Application Diskette Builder (ADB) is a Professional application that creates a master copy of an application that can be reproduced for distribution. It uses the information in your application command (.INS) file to copy your application from the hard disk to one or more diskettes, from which it can be installed on other systems. It is documented in the Tool Kit Reference Manual.

2.2.3 Communications

Communications services allow you to perform communications operations on the Professional. They fall into three categories:

SOFTWARE DEVELOPMENT TOOLS

1. Base System Services are part of the P/OS operating system. They include an asynchronous driver (XKDRV) for the communications port, as well as a communications service library (COMLIB). The Base System Services allow you to set up and control a telephone connection for data communication and to handle voice communication if you install the optional Telephone Management System (TMS). These services are documented in the Tool Kit Reference Manual and the P/OS System Reference Manual.
2. PRO/Communications Services consists of a set of routines that allow your application to access phone book, communications setup, and file transfer utilities as well as the Terminal Emulator. These services require that the end user has installed the PRO/Communications application on the target system. These services are documented in the Tool Kit Reference Manual and the PRO/Communications Manual.
3. Telephone-Management System (TMS) Services consist of a set of routines that allow your application to control the TMS hardware. These services require that the end user has installed the TMS application and hardware on the target system. These routines are documented in the Tool Kit Reference Manual and the Telephone Management System (TMS) Programmer's Manual.

2.2.4 CORE Graphics Library (CGL)

The CORE Graphics Library (CGL) is a general-purpose graphics subroutine package based on the ACM SIGGRAPH CORE Standard, with additional instructions that provide high-level access to the Professional 300 series video bitmap. Features include:

- Automatic mapping of user-defined coordinates to graphics devices.
- Lines, curves, polygons, rectangles, solid objects, and graphics text.
- Control of styles, textures, and colors.
- Multiple user-defined fonts.
- Support for graphics plotters.
- Easy access from high-level languages.
- Device independence and transportability.

The CORE Graphics Library is documented in the CORE Graphics Library Manual.

SOFTWARE DEVELOPMENT TOOLS

2.2.5 DECnet

PRO/DECnet software allows Professional 350 computers to connect to other DECnet systems on the Ethernet. PRO/DECnet is an end node only implementation of the Phase IV Digital Network Architecture. It is compatible with other Phase III and Phase IV DECnet products. The PRO/DECnet software supports:

- Multiple, simultaneous logical links between a Professional 350 and any other Phase III or Phase IV DECnet system.
- Task-to-task communication between a Professional 350 and any other Phase III or Phase IV DECnet system.
- High speed resource sharing within a local area network.
- Various network management and maintenance functions.
- Transport facilities that permit programs utilizing RMS-11 V2.0 to access remote files.

The Tool Kit documentation set includes a volume devoted to PRO/DECnet.

2.2.6 General Image Display Instruction Set (GIDIS)

PRO/GIDIS is a general-purpose graphics subroutine package that provides low-level virtual device access to the Professional video bitmap. Use PRO/GIDIS when speed and compactness are of primary importance; for example, with systems software such as:

- interactive drawing packages
- graphics terminal emulators
- scientific/engineering data displays
- rapid picture display programs

The application interface for PRO/GIDIS is the RSX QIO call. It is intended to be used from MACRO-11.

PRO/GIDIS is documented in the PRO/GIDIS Manual, part of the PRO/Tool Kit documentation set. See that manual for a more detailed comparison of CGL and PRO/GIDIS.

SOFTWARE DEVELOPMENT TOOLS

2.2.7 Fast Install

Fast Install is a Professional 350 application that allows you to install an application from a directory on the hard disk. (P/OS Disk/Diskette Services can install an application from diskette only.) Fast Install is distributed with the Host Tool Kit and the PRO/Tool Kit on the Application Diskette Builder (ADB) diskette. It is documented in the Tool Kit Reference Manual.

2.2.8 File Control Services (FCS-11)

File Control Services (FCS) is a set of file management routines for use on the RSX-11 family of operating systems. FCS is included in the Tool Kit only for the purpose of converting existing applications to P/OS. You should use PRO/RMS-11 when you write new applications. FCS is described in the IAS/RSX-11 I/O Operations Reference Manual, which is included in the RSX-11M or RSX-11M-PLUS documentation set (not included with the Tool Kit).

2.2.9 Forms Management System (PRO/FMS-11)

The Forms Management System (PRO/FMS-11) is the Tool Kit version of FMS-11 for PDP-11 systems. FMS-11 performs screen management and input validation by providing your application with access to a set of predefined forms. A form can contain display-only information as well as one or more fields, each of which has a set of help, display, protection, and validation characteristics. PRO/FMS-11 is made up of three software components:

- The Form Editor (PROFED) is an editor for creating and modifying video forms. You edit forms while viewing them on the screen as they are to appear to the application user. You can modify existing forms without having to recompile the application or reprocess collected data.
- The Form Utility (PROFUT) is a utility for creating and maintaining binary form library files. If your target system is P/OS Hard Disk, you can use these form libraries to reduce program memory requirements.
- The Form Driver (FDV) is an object module containing a set of callable subroutines that display forms, perform input and output operations, respond to HELP requests, and so forth. The Form Driver calls for the Professional produce somewhat different results from FMS-11 calls for the VT100 terminal.

SOFTWARE DEVELOPMENT TOOLS

The PRO/FMS-11 binder (included with the Tool Kit documentation set) provides manuals that describe this tool in detail.

2.2.10 Frame Development Tool (FDT)

The Frame Development Tool (FDT) creates menu, help, and message frames. FDT is used by application developers to create frames through which an end user interacts with an application. Through a series of forms, you specify the actual text to be displayed on menus and help frames and the information relating to the manner and timing of the frame displays. Service routines in POSRES must be called to display the frames. The Frame Development Tool is documented in the Tool Kit Reference Manual.

2.2.11 On-Line Debugging Tool (ODT)

The On-Line Debugging Tool (ODT) is a utility for debugging task images. You can use it to control program execution, display and alter the contents of memory locations and registers, search and fill memory, and perform calculations. ODT is provided for debugging MACRO programs, and programs written in higher level languages that do not provide their own debuggers. It is documented in the IAS/RSX-11 ODT Reference Manual and supplement.

2.2.12 Print Services Callable Task

Print services consists of a routine that allows your application to print a file, stop, continue, abandon or restart a print job, or obtain printer status. A request to print a file creates a non-interactive task. Print services is documented in the Tool Kit Reference Manual.

2.2.13 POSRES User Interface Services Library

The P/OS User Interface Services Library (POSRES) consists of a set of routines that allow your task to have a menu-based user interface consistent with that used by P/OS. These routines open and close frame definition files, pack, unpack, read, and display menus, help frames, and message frames, invoke the P/OS "New File" and "Old File" frames, and process function keys. The Tool Kit User's Guide contains a description of how to design and implement a menu-based user interface. The POSRES routines are documented in the Tool Kit Reference Manual.

SOFTWARE DEVELOPMENT TOOLS

2.2.14 POSSUM System Services Library

The POSSUM System Services Library consists of a set of routines that allow your task to manipulate file attributes, directories, volumes, logical names, tasks, regions, and commons. POSSUM is documented in the P/OS System Reference Manual.

2.2.15 PROSE Callable Editor Task

The PROSE callable editor task (CET) allows your application to offer PROSE, the text editor supplied with P/OS, for use within its own context. PROSE offers facilities for entering and editing text to create documents, source programs, and memos or similar text files. Editing keys on the Professional keyboard allow text manipulation. The end user documentation describes PROSE and the PROSE user interface. The callable editor task is documented in the Tool Kit Reference Manual.

2.2.16 Record Management Services (PRO/RMS-11)

Record Management Services (PRO/RMS-11) provides an interface between the Professional's file system and your application. All of the Tool Kit high-level languages include built-in support for PRO/RMS-11. Thus, unless you are programming in MACRO-11 or have I/O requirements beyond those provided by your language, you need not be concerned with PRO/RMS-11.

PRO/RMS-11 includes a set of run-time service routines that enable direct, sequential, and multi-keyed access to data files. The routines also let your program define, populate, update, and maintain files on direct access devices.

The symbol tables, object module libraries, and overlay descriptor files for task building P/OS applications against the PRO/RMS-11 resident library are included with the Tool Kit, as well as a PRO/RMS-11 Macro library. The PRO/RMS-11 binder (included with the Tool Kit documentation set) provides manuals that describe this tool in detail.

2.2.17 SORT Callable Sort Task

PRO/SORT is a general-purpose sorting utility that is callable from your applications. PRO/SORT is documented in the Tool Kit Reference Manual.

CHAPTER 3

THE PROFESSIONAL 300 SERIES PERSONAL COMPUTER

3.1 HARDWARE CONFIGURATIONS

The Professional 300 Series of personal computers has two hardware configurations: the Professional 325 and the Professional 350. These differ in their ability to handle optional equipment and software. The basic system consists of a system unit, keyboard, video monitor, and storage media:

- The system unit houses a 16-bit PDP-11 minicomputer. At the base of the Professional is a main system logic module, or system board, based on the PDP-11 Central Processing Unit (CPU). It supports a virtual address space of 64 K bytes. Its standard configuration provides 512 K bytes of random access memory. Additional memory can be added in 256 Kb increments.
- The keyboard has four distinct regions: function keys, numeric keypad, text-editing keys and traditional keyboard (see Figure 3-1). These work with P/OS to create a clear and easy interface between your application and the end user. When your application is running, it can redefine the use of all but the system function keys.
- The standard video monitor (VR201) has a 12"-diagonal, low-reflection screen, with tilt adjustment in the back.
- The Professional has two kinds of storage media: diskettes and the hard disk. Every system unit comes with a dual diskette drive that provides over 800 Kb of storage capacity on two 5-1/4" diskettes. The Professional 350 can also support an optional 5 Mb hard disk (RD50) or 10 Mb hard disk (RD51).

3.2 THE PROFESSIONAL OPERATING SYSTEM (P/OS)

All Professional 300 systems come equipped with the Professional Operating System, P/OS. Based upon a pre-built, single-user version

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

of the RSX-11M-PLUS operating system, P/OS employs many of the same methods of managing resources as the RSX family of software. For example, checkpointing temporarily removes lower-priority tasks from memory so that higher-priority tasks can run.

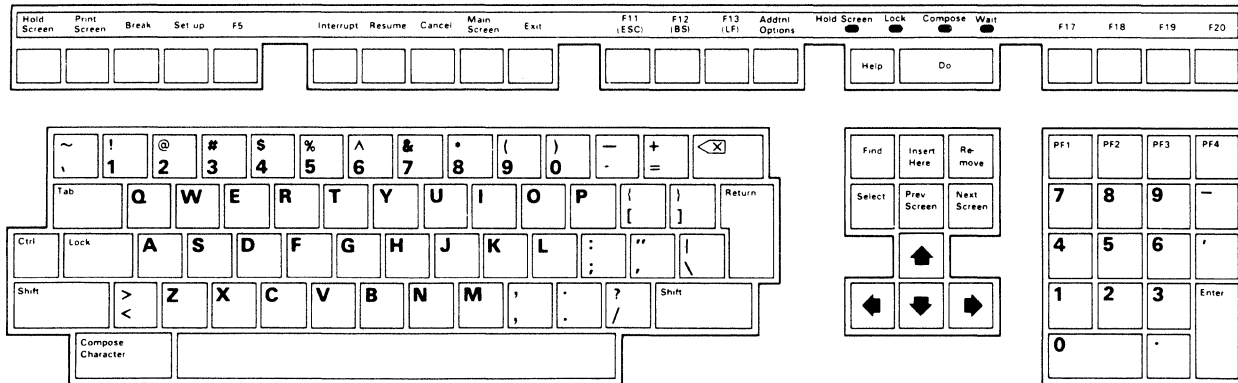


Figure 3-1: Professional Keyboard (U.S./Canada)

To support the two hardware configurations, P/OS also has two configurations: P/OS Hard Disk and P/OS Diskette:

- P/OS Hard Disk runs only on a Professional 350 with the hard disk. Once copied onto the hard disk from the distribution diskettes, its system software resides on hard disk and is automatically loaded into memory during system use.
- P/OS Diskette, a subset of P/OS Hard Disk, supports the entry-level systems in the Professional family: the Professional 325 and the Professional 350 without hard disk. P/OS Diskette software resides on the distribution diskettes and is loaded into memory each time the Professional is powered up.

P/OS Diskette can also be used on a Professional 350 with hard disk, where it will run as if the hard disk were not present. Both the Professional 325 and 350 diskette systems can upgrade to a Professional 350 Hard Disk. The Professional computer can therefore consist of any of the following configurations:

- the Professional 325, running P/OS Diskette
- the Professional 350 without hard disk, running P/OS Diskette
- the Professional 350 with hard disk, running P/OS Diskette

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

- the Professional 350 with hard disk, running P/OS Hard Disk

The Professional end user is provided with a full set of documents describing the Professional and the use of P/OS services. You have these manuals as part of your Tool Kit; refer to them for detailed instructions on operating the Professional.

3.2.1 The P/OS User Interface

The P/OS user interface consists of menus, messages, forms, on-line help, and function keys:

- **Menus**

The end user operates the Professional by making choices from menus (lists of options) that appear on the screen. To make a selection, the user moves the pointer to a menu item or types an item on a command line, and presses the DO key.

- **Messages**

The Message/Status board displays information about errors or completed activities without disturbing the current process.

- **Forms**

The user provides information to an application or system service by filling in blanks on easy-to-understand forms that appear on the screen.

- **Function keys**

The user can execute many operations by pressing function keys on the top row of the keyboard and on the numeric keypad.

- **On-line help**

P/OS comes with its own context-sensitive help, which the user can access at any point by pressing the HELP key.

3.2.2 P/OS Services

P/OS includes File, Disk/Diskette, Print and text editing services. P/OS users will be familiar with these as the primary means of handling data on the Professional. Your application also can access these services.

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

- **File services** manipulate files on disk or diskette, allowing the user to copy, rename, delete, display and delete-protect files. P/OS Hard Disk file services also allow the user to back up and restore files on disk.
- **Disk/Diskette services** allow the user to initialize diskettes; to install, remove, start and stop applications; to unlock a file; to copy the contents of one diskette to another; to create, delete, and list directories; to display the current directory, and, in some cases, to change the current directory. P/OS Hard Disk services allow the user to integrate applications into and to remove them from the Main Menu.
- **Print services** print stored files on a printer. Print services start a print job, pause and continue it, restart printing at the beginning of a file, abandon the print job, view the status of the printer, and set printer characteristics.
- **PROSE** is the P/OS text editor. It is suitable for writing memos or entering data. (PROSE is described in detail in the Professional 300 Series User's Guides.)
- **PRO/Communications** includes the Terminal Emulator program and File Transfer. The Terminal Emulator program allows the Professional to operate as a VT52, VT102, or a VT125 terminal connected to a host. The File Transfer service allows files to be passed between the Professional and a host system, and is used as part of application development with the Host Tool Kit. (See the PRO/Communications Manual for full details.)

3.2.3 P/OS System Components

P/OS consists of a number of modules, some of which are resident in memory at all times, some of which are brought into memory only as needed by the application. These modules are packaged as part of P/OS (there is no system generation). This section briefly describes the main P/OS system components: the P/OS Executive, PRO/Dispatcher, and cluster libraries.

3.2.3.1 The P/OS Executive

The Executive is the "nucleus" of the Professional Operating System. It manages system resources such as memory and the CPU and serves as the software interface to the Professional hardware. The P/OS Executive was derived from the RSX-11M-PLUS Version 2.0 executive. It was modified to fit into a single-user personal computer environment. The P/OS System Reference Manual describes the P/OS Executive.

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

The P/OS Executive is essentially the same for P/OS Hard Disk and P/OS Diskette. It performs the following functions:

- Provides an intertask communication mechanism to the PRO/Dispatcher, thus allowing the PRO/Dispatcher to carry out its application-related jobs.
- Services executive directives that are accessible through macro calls. Executive directives may be used by applications to obtain system information, such as the date and time, and to control application execution, such as for multitasking.
- Provides access through device drivers to the dual diskette drive and, with P/OS Hard Disk, the hard disk. When a diskette is inserted in a drive, P/OS Executive automatically adds the diskette name to the list of available volumes. When the diskette is removed, the name is removed.

The end user is not required to perform device mount and dismount. Disk/Diskette services provide the end user with access to the media.

- Manages disk/diskette storage through PRO/RMS-11 and the on-disk or on-diskette file structure. File, Disk/Diskette, and Print services provide those facilities to the end user.
- Provides access through the terminal driver to the terminal subsystem. The terminal subsystem controls the video image and the keyboard. Applications can use the terminal subsystem services along with the CORE Graphics library and PRO/GIDIS to display images on the screen and interpret keyboard input.
- Provides access through the terminal driver to the printer. The PRINT SCREEN key and Print Services allow the end user to print on the printer.

3.2.3.2 PRO/Dispatcher

This is the primary, menu-oriented interface between the end user and P/OS. The ProDispatcher performs many of the same functions as a command language interpreter does on more traditional computers. The PRO/Dispatcher controls the user interface in the following ways:

- Integrates the application software with P/OS. With P/OS Hard Disk, adds the name of the application to the user-specified menu when the end user runs Install. With Diskette P/OS, copies all P/OS system software required to run the application to the application diskette when the end user runs Prepare Application.

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

- Adds the name of the application tasks to the list of executable tasks that make up the System Task Directory (STD).
- Invokes, aborts, and removes tasks. Uses the application installation file to start the first task for that application.
- With P/OS Diskette, loads the communications port driver or installs the graphics support system, if either is requested in the application installation file. Removes them when the application exits.
- Opens any default menu and help files specified in the application installation file.
- If the end user presses the INTERRUPT key followed by the DO key, PRO/Dispatcher aborts all tasks of the running application and, after a specific period of time, removes all application tasks and commons.

NOTE

INTERRUPT/DO is equivalent to CTRL/C. To prevent application termination after the INTERRUPT/DO sequence, the task can field the sequence by attaching the terminal for CTRL/C ASTs. See the high-level language documentation for information about how to attach the terminal for CTRL/C ASTs. In MACRO-11, read-pass-all or attaching to the terminal for CTRL/C ASTs can be used to trap CTRL/C. See the P/OS System Reference Manual for more information.

- Recovers the system after disabling errors have occurred in a running application. If a task fails, control returns to PRO/Dispatcher, and the Main Menu (P/OS Hard Disk) or other system message (P/OS Diskette) is displayed.

3.2.3.3 Cluster Libraries

Cluster libraries provide a single copy of commonly used routines in a way that can be shared by several tasks. Pre-built as an integral part of the operating system, cluster libraries share the same virtual address space and are loaded into physical memory as needed. The P/OS cluster libraries are:

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

- Programming language object-time or run-time systems
- PRO/RMS-11 (RMSRES)
- CORE Graphics Library (CGLFPU)
- P/OS Services Library (POSRES)
- PRO/Communications (COMLIB)
- P/OS System Services library (POSSUM)

(For more information, see Chapter 7 and Chapter 5).

3.2.4 Calling P/OS Routines from High-level Languages

P/OS routines use the standard PDP-11 R5 calling sequence convention (sometimes called the FORTRAN Calling Sequence Convention). This section provides some general information about the R5 convention. The subsequent sections describe how to call P/OS routines from each high-level language.

This manual and the Tool Kit Reference Manual provide a "Format:" description for each P/OS routine that shows the external routine name followed by a parameter block. The R5 calling sequence convention requires that you pass all parameters by reference. In other words, the parameter block contains only addresses of parameters, not actual data.

The data type and relative position of each parameter must match that expected by the P/OS routine. Assume that there is little or no data type checking of parameters. If a routine doesn't work correctly, check the parameter data types. One of the most common bugs is the specification of a real (floating point) parameter where an integer is required.

Some languages allow you to pass an expression as a reference parameter. The language's run-time library evaluates the expression, stores it in a temporary location, and passes the address of the location. If your language does not support this, read "expression" as "constant or variable."

You can use arrays for multi-word parameters. For example, you can use a two-word integer array for the POSRES status block. You must, however, know how your language numbers arrays. For example, BASIC-PLUS-2 numbers all arrays from zero, while PASCAL allows you to specify your own numbering scheme.

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

3.2.4.1 BASIC-PLUS-2

In BASIC-PLUS-2, external subprogram calls do not have to be declared. A call has the format:

CALL name BY REF (p1, p2, ..., pn)

name is the name of the external subprogram.

BY REF specifies that the parameters are to be passed by reference.

p1,p2,... are actual parameters as described.

Refer to your BASIC-PLUS-2 documentation for detailed information on calling external routines from BASIC-PLUS-2.

Notes:

- To pass an array, include the (empty) parentheses in the BASIC-PLUS-2 call.
- BASIC-PLUS-2 does not allow you to pass array elements by reference.
- You can pass a dynamic string variable, using the LEN function to determine it's length. For example:

CALL name BY REF (... , S\$, LEN(S\$), ...)

3.2.4.2 COBOL-81

In COBOL-81, external routine calls do not have to be declared. A call has the format:

CALL "name" USING p1 p2 ... pn.

name is the name of the external routine.

p1 p2 ... are actual parameters as described.

Refer to the Tool Kit COBOL-81 Documentation Supplement for detailed information on calling P/OS routines from COBOL-81.

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

3.2.4.3 DIBOL

In DIBOL, external subroutine calls do not have to be declared. A call has the format:

XCALL name (p1, p2, ..., pn)

name is the name of the external subroutine.

p1,p2,... are actual parameters as described.

Refer to the Tool Kit DIBOL User's Guide for detailed information on calling P/OS routines from DIBOL.

3.2.4.4 FORTRAN-77

In FORTRAN-77, external subroutine calls do not have to be declared. A call has the format:

CALL name (p1, p2, ..., pn)

name is the name of the external subroutine.

p1,p2,... are actual parameters as described.

Refer to the Tool Kit FORTRAN-77 Documentation Supplement for detailed information on calling P/OS routines from FORTRAN.

3.2.4.5 PASCAL

In PASCAL, an external procedure declaration has the format:

PROCEDURE name (VAR p1; VAR p2; ... VAR pn); SEQ11;

name is the name of the external routine.

VAR specifies pass by reference.

p1;p2;... are formal parameters as described.

SEQ11 specifies the PDP-11 R5 calling sequence.

A procedure call has the format:

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

name (p1, p2, ..., pn);

name is the name of the external routine.

p1,p2,... are actual parameters that match the formal parameters in the procedure declaration.

Refer to the Tool Kit PASCAL User's Guide for detailed information on calling P/OS routines from PASCAL.

Notes:

- You can declare formal parameters with the READONLY attribute so that you can pass constants as actual parameters.
- You can declare formal string (array [1..n] of char) parameters with the UNSAFE attribute so that you can pass strings of different lengths as actual parameters.

3.2.5 Calling P/OS Routines from MACRO-11

To transfer control to a P/OS routine:

JSR PC, name

name is the name (global entry point) of the routine.

General purpose register 5 (R5) contains the address of the parameter block which has the following format:

high byte	low byte
0	number of parameters
address of parameter 1	
address of parameter 2	
.	
.	
.	
address of parameter n	

When the P/OS routine returns, the contents of registers R0 through R5

THE PROFESSIONAL OPERATING SYSTEM (P/OS)

are undefined. The stack pointer (SP) is restored to its state at routine entry.

CHAPTER 4

THE APPLICATION DEVELOPMENT CYCLE

4.1 THE DESIGN PHASE

Designing software for the Professional 300 series requires consideration of several factors.

- target system configurations
- virtual address space
- physical memory
- user interface

These factors will affect all other development phases; thus, it's important to resolve them before beginning implementation. The first three are discussed in detail in Chapter 5. User interface design is covered in Chapter 8.

4.2 THE IMPLEMENTATION PHASE

Implementation consists of creating the following:

- **Application Directory**

To test your application on P/OS Hard Disk, you must create a directory to contain the executable files and installation command (.INS) file. To create the directory, use the DCL CREATE/DIRECTORY command or P/OS Disk Services.

NOTE

The application directory must have the same name as the .INS file. For example, if the installation file is PROGRAM.INS, the directory must be named PROGRAM.

THE IMPLEMENTATION PHASE

If you are working with the PRO/Tool Kit, use this directory to contain all of your application files.

- **Program Source Code**

Use any text editor to create your source code. If you are using the PRO/Tool Kit, you can use the EDIT command to invoke EDT (the default) or PROSE. If you use PROSE, do not save word/wrap/margin settings.

- **Frame and Form Files**

Use the Frame Development Tool (FDT) (described in the Tool Kit Reference Manual) or the Forms Editor (FED) (described in the FMS-11/RSX Software Reference Manual) to create the frame and form files associated with your application. Sketch out your frames and forms before creating them and keep hard-copy descriptions on hand while working on your source code.

- **Professional Application Builder (PAB) Files**

Use any text editor to create the PAB command and/or .ODL files for building a task image, as described in Chapter 7. Most of the Tool Kit programming languages have a facility for generating PAB files that you can tailor to your application. Some of the tools, (CGL, POSRES, PRO/FMS-11, and so forth) require that you make some changes to the PAB files.

- **Installation Command Files**

One application installation command (.INS) file is required for each Professional 300 series target configuration that your application supports. The format and contents are described in the Tool Kit Reference Manual.

4.3 THE BUILD PHASE

You may go through this procedure many times while developing an application.

1. **Compile or assemble the source code.**

Use your language processor to create the object modules (.OBJ files) that will make up your task images. For convenience, write a command procedure that compiles or assembles all of the files automatically. If you are working with the PRO/Tool Kit, use the SPAWN command to invoke it. If you are working on a host system, use the batch queue.

THE BUILD PHASE

2. Build the task images.

Use the Professional Application Builder (PAB) to create task images that can be executed on P/OS. PAB is described in Chapter 7. You can create a command procedure that invokes PAB and use it in a noninteractive mode as described above.

3. Prepare frame and form files.

If your application uses frame description files, run FDT and use the CONVERT command to create .HLP, .MNU., and .MSG files in executable format, as described in the Tool Kit Reference Manual. If your application uses form files, run the Forms Utility (FUT) and create form libraries as described in the FMS-11/RSX Software Reference Manual.

4.4 THE TESTING PHASE

If you are using the PRO/Tool Kit, please turn to Section 4.4.1. If you are using the host Tool Kit, please turn to Section 4.4.2.

4.4.1 Testing Applications on the PRO/Tool Kit

If you are using the PRO/Tool Kit, you can run your application from DCL command level or from the P/OS User Interface. This section describes how to run your application from DCL. If you want to run from P/OS, exit from the PRO/Tool Kit and turn to Section 4.4.2, Step 2.

NOTE

This procedure assumes that you have read Section 6.3.2. If not, please do so before proceeding.

1. Set the P/OS current directory to your application directory.
Type:

```
$ SET DEF [name]
```

where "name" is the name of your application directory.

2. Install the libraries used your application. For example:

```
$ INSTALL [ZZSYS]CGLFPU/READONLY
```


THE TESTING PHASE

3. Run the application. Type:

\$ RUN TEST

4.4.2 Testing Applications on the Host Tool Kit

To test run your application with the host Tool Kit, follow these steps:

1. Transfer the files to the target system.

Invoke the Professional Terminal Emulator, use the Professional File Transfer utility to copy your application files to the application directory on your Professional, and exit from the Terminal Emulator. (See the PRO/Communications Manual for details.)

NOTE

For the second and subsequent test runs, you need transfer only files that have changed since the last test. P/OS uses the latest versions of files by default. (You can specify version numbers in the installation command file to override the default.)

2. Install the application.

Use the Fast Install utility described in the Tool Kit Reference Manual.

NOTE

Once you have installed your application, it is generally unnecessary to reinstall it when you copy new files to your application directory.

3. Run the application.

Select the application from the menu on which you installed it. If it won't start up or aborts, check the P/OS error codes listed in the Tool Kit Reference Manual. Also, make sure that the installation command file installs all necessary tasks.

THE DEBUGGING PHASE

4.5 THE DEBUGGING PHASE

Some high-level languages provide interactive debugging software that allows you to control program execution, manipulate variables, and so forth. Refer to your language documentation for more information.

The Tool Kit allows you to connect a separate terminal to your Professional in order to control the debugger. This is especially useful when debugging programs that use the Professional video screen: menu interfaces, graphics, and so forth.

Use the console cable provided with the Tool Kit to attach your choice of debugging terminal (any of the VT100 family of terminals, an LA120, or LA12) to the printer port on the back of the Professional system unit.

NOTE

The console cable is for debugging only. If you want to use an LA50, LQP02, or LA100 printer for normal printing purposes, use a printer cable.

Set the terminal to run at 9600 baud. You can then redirect debugger I/O to the debugging terminal, using language-specific debugging commands, while program I/O continues to display undisturbed at the Professional.

4.6 THE TUNING PHASE

There are several techniques that you can use to improve the performance of your application. They are discussed in detail in Chapter 9.

NOTE

Most of the tuning information is intended for experienced RSX-11M/M-PLUS programmers.

4.7 THE DISTRIBUTION PHASE

When your application is debugged and ready to distribute:

- Create a distribution kit.

THE DISTRIBUTION PHASE

Use the Application Diskette Builder (ADB) (described in the Tool Kit Reference Manual) to create a master application distribution kit on one or more diskettes.

- Test the distribution kit.

To test your distribution kit, use P/OS services to install and run your application on all supported Professional hardware configurations.

CHAPTER 5

APPLICATION DESIGN CONSIDERATIONS

5.1 TARGET SYSTEM CONFIGURATIONS

As you develop your application, you must decide whether it will run on P/OS Hard Disk, P/OS Diskette, or both. Ideally, an application will run on all Professional configurations. Greater storage capacities are available to programs running on P/OS Hard Disk, as well as greater speed. Although you can design an application that runs on both P/OS Hard Disk and P/OS Diskette, certain applications well-suited for hard disk systems may be unsatisfactory or even unsuccessful when used with P/OS Diskette. You should consider your audience and your performance requirements when you choose your target systems.

The following differences between P/OS Hard Disk and P/OS Diskette can affect your design:

- **Device Speed**

The hard disk drive is a much faster device than the diskette drive. Applications which are disk-intensive may perform poorly on diskette-based systems. These include applications that are disk-overlaid, applications that consist of multiple task images and therefore may require checkpointing, and applications that call several disk-resident P/OS cluster libraries and therefore cause library swapping between memory and diskette.

Any application that uses diskettes heavily will run more slowly on P/OS Diskette than it would on P/OS Hard Disk and may not be suitable for diskette-based systems.

- **Mass Storage**

A P/OS Diskette application must fit on one diskette. Unlike the distribution diskette for Hard Disk P/OS, a distribution diskette for P/OS Diskette must include files which would otherwise be on the hard disk. These include frame definition and cluster library files for language support and additional task files that may be

TARGET SYSTEM CONFIGURATIONS

required (for example, communications, print server, or callable editor tasks). Also, because P/OS Diskette copies system software to the application diskette, you must leave room on the diskette for these files. See Appendix B for details on diskette space requirements.

Because a P/OS Diskette application uses one diskette drive while running, it can access only one data diskette. P/OS Hard Disk applications can access two data diskettes.

To calculate diskette usage by application, add the total file sizes in the application directory to the number of components used by the application. The file sizes are shown in Appendix B. Also add the size of the checkpoint file, if any. If the total exceeds approximately 745 blocks, the application may be too big to fit on one diskette.

● Print Services

With Hard Disk P/OS, callable Print Services run in the background while the application goes on to other processing. With Diskette P/OS, Print Services returns control to the application only after printing is complete (see the Tool Kit Reference Manual). Applications that use printing may not be suitable for diskette-based systems.

5.2 VIRTUAL ADDRESS SPACE

User tasks on the Professional are limited to a virtual address space of 32K words. The task may map to other regions in memory but at any given instant only 32K words of memory may be referenced. This places a constraint on the amounts and locations of code and data that a task may contain. There are several options that you can exercise should your task become so large that 32K words is no longer sufficient. They are described in the following sections.

5.2.1 Overlaying

One of the traditional methods for most machines is to overlay the task image such that only certain portions are addressable at a time. This method will work for tasks that are structured such that portions of the task can be logically separated. Overlaying is discussed in:

VIRTUAL ADDRESS SPACE

- Chapter 7 of this manual.
- The RSX-11M/M-PLUS Task Builder Manual.
- The POS System Reference Manual.
- The Guide to Writing a Device Driver and Advanced Programmer's Notes.

5.2.2 Cluster Libraries

Cluster libraries share the same virtual address space and bear similarities to memory-resident overlays for your task. As long as these cluster libraries do not reference each other there are no complications, and your task gains in the amount of virtual address space that it may use. Cluster libraries are discussed in:

- Chapter 7 of this manual.
- The POS System Reference Manual.
- The RSX-11M/M-PLUS Task Builder Manual.
- The Guide to Writing a Device Driver and Advanced Programmer's Notes.

5.2.3 Multiple Tasks

Another option to consider is an application consisting of multiple tasks. This is only possible if the application can be separated into functionally independent units. It is possible to transfer data between the tasks comprising the application, but the cost of this can be high. A carefully written multi-task application can perform extremely well and is an option that should be considered. This topic is discussed in the P/OS System Reference Manual.

5.2.4 Multiple Regions

If the task needs more virtual address space for data storage then a possibility that emerges is to create a separate data region for the task. Data may be stored in this region and kept outside of the task's virtual address space until it is needed. At that time the region is mapped by the task, data is manipulated and then the task

VIRTUAL ADDRESS SPACE

unmaps the region to continue working somewhere else. This topic is discussed in the P/OS System Reference Manual.

5.3 PHYSICAL MEMORY

You must consider not only the amount of virtual address space available to your application, but also the physical memory constraints of the system. It is very possible to create an application that runs slowly (or not at all) due to memory contention or deadlocks. These can result from either ignoring the issue of physical memory or overestimating its size.

On a standard Professional 300 system there are 512K bytes of memory. Of this slightly less than one-half, approximately 240K bytes, is available for use by the application. Into this memory must fit all tasks, data regions, libraries and drivers required concurrently by the application. If the application task exceeds this amount, checkpointing will result.

5.3.1 Checkpointing

Checkpointing will start to occur as the amount of memory used by an application approaches 240K bytes. Checkpointing is not necessarily bad, it is merely a way of relieving the pressure of memory contention caused by too many requests for too little memory. There are some methods by which the impact of checkpointing on an application can be minimized. These include using read-only or multi-user regions; they are cheaper than read-write regions as they need not be written back out to the disk.

CHAPTER 6

USING APPLICATION FILES

6.1 P/OS FILE SPECIFICATIONS

A P/OS file specification is a character string that identifies a unique device, directory, or file. It consists of one or more fields, separated by punctuation marks. Each field is explained in detail in subsequent sections of this chapter.

A file specification has the format:

```
device:[directory]filename.typ;version
```

device	an alphanumeric string that specifies the name of a peripheral device.
directory	an alphanumeric string that specifies the name of a directory.
filename	an alphanumeric string that specifies the name of a file.
typ	an alphanumeric string that specifies the type of a file.
version	a numeric string that specifies the version number of a file.

You can omit fields from a file specification and let the system provide default values. These defaults are discussed in detail in the following sections.

P/OS FILE SPECIFICATIONS

Examples

DWl:	(device)
DWl:[USERFILES]	(device and directory)
DWl:[USERFILES]TEST	(device, directory, and name)
DWl:[USERFILES]TEST.DAT	(device, directory, name, and type)
DWl:[USERFILES]TEST.DAT;7	(device, directory, name, type, and versic
[USERFILES]TEST.DAT;7	(directory, name, type, and version)
TEST.DAT;7	(name, type, and version)

Notes

- Directory names, file names, file types, and version numbers apply only to files on mass storage (file-structured) devices such as disks and diskettes.
- The maximum length of a file name is 9 characters.
- The maximum length of a file type is 3 characters.
- File specifications are not case-sensitive. For example, these are identical:

DWl:[USERFILES]TEST.DAT;7 dwl:[userfiles]test.dat;7

- VAX/VMS and RSX-11M/M-PLUS file specifications allow square brackets ([]) or angle brackets (<>) to enclose a directory name field. For example:

[USERFILES] is equivalent to <USERFILES>

With P/OS, you must use square brackets.

- VAX/VMS file specifications allow a semicolon (;) or a period (.) to separate the file type field from the version number field. For example:

TEST.DAT;7 is equivalent to TEST.DAT.7

With P/OS, you must use a semicolon.

6.1.1 Devices

Each peripheral device known to P/OS has a permanent name called a physical device name. It has the format:

ddnnn:

P/OS FILE SPECIFICATIONS

dd a two-character alphabetic string that specifies the device type.

nnn an optional one- to three-digit numeric string that specifies a particular device, or, if a device has more than one drive, a particular drive.

Some devices exist in name only. All I/O requests for a "pseudo-device" are redirected to another physical device. This provides device independence for standard naming conventions. Table 6-1 shows the P/OS physical device names and pseudo-device names.

Table 6-1: P/OS Physical and Pseudo Device Names

<u>Device</u>	<u>Physical Names</u>
RD50 Hard Disk	DW1
RD51 Hard Disk	DW1
RX50 Diskette	DZ1, DZ2
Keyboard/Monitor	TT1
Printer Port	TT2
Comm Port	XK
<u>Device</u>	<u>Pseudo Names</u>
System library	LB
Printer port	LP
User default	SY
Keyboard/monitor	TI

Notes

- You can use a logical name to specify a peripheral device. See Section 6.2 for information about logical names.
- If a file specification contains no device name, P/OS provides one by default. The default device has the pseudo-device name SY:. When the end user or the application changes the default device, P/OS reassigns the pseudo-device name SY: to the new device. Always use the name SY: in file specifications unless you are trying to access a specific device by name.

P/OS FILE SPECIFICATIONS

- If you are using P/OS Hard Disk, you can specify the default device with either File Services, Disk/diskette services, or SET-UP MENU A. The last is a "permanent" change: it stores the new default device so that will take effect whenever you start the system. When P/OS Hard Disk starts up the first time, the default device is BIGVOLUME:.
- If you are using PRO/Tool Kit DCL, you can specify the default device with the SET DEFAULT command.
- If you are using P/OS Diskette, you can specify the default device with File Services. The system resets the default device to USERDISK: each time an application starts up. USERDISK: translates to the volume name of the diskette, if any, mounted in drive DZ2.
- An application program can specify the default device by calling PROLOG from the POSSUM library (see the P/OS System Reference Manual).

6.1.2 Directories

A directory is a file that identifies a set of files on a disk or a diskette. Most people prefer to think of a directory as a named set of files on a disk or diskette. That a directory is itself a file has little to do with its function. Thus, we will use the terms "directory" and "directory file" to mean a set of files and the file that identifies them, respectively.

A file's name, type, and version number uniquely identifies that file within a directory. Different files with the same name, type, and version number can exist in other directories.

In a file specification, square brackets ([]) or angle brackets (<>) indicate that the contents are a directory name. Directory names can have the following formats:

- A one- through nine-character alphanumeric string. For example:

[PROGRAMS] <INVENTORY> [RECIPES]

- A two-part octal number in the format of a user identification code (UIC). Separate the group number from the member number with a comma. For example:

[0,0] [1,5] [150,13] [240,222]

P/OS FILE SPECIFICATIONS

- A six-character numeric string in UIC format. Omit the comma and specify right-justified (zero-filled) numbers. For example:

[000000]

[001005]

[150013]

[240222]

Notes

- You can use a logical name to refer to a directory in some instances. See Section 6.2 for information about logical names.
- If a file specification contains no directory name, P/OS provides one by default. A pair of empty square brackets ([]) is an explicit request for the default directory.
 - If you are using P/OS Hard Disk, you can specify the default directory with either File Services, Disk/diskette services, or SET-UP MENU A. The last is a "permanent" change: it stores the new default directory so that will take effect whenever you start the system. When P/OS Hard Disk starts up the first time, the default directory is [USERFILES].
 - If you are using PRO/Tool Kit DCL, you can specify the default directory with the SET DEFAULT command.
 - If you are using P/OS Diskette, you can specify the default directory with File Services. The system resets the default directory to [USERFILES] each time an application starts up.
 - An application program can specify the default directory by calling PROLOG from the POSSUM library (see the P/OS System Reference Manual).
- Numbered directories and directories that begin with the letters "ZZ" are reserved for system software. Directories that begin with the letters "ZZAP" contain installed applications. File Services and Disk/diskette Services do not display these directories unless you specifically ask for them.
- Your application can use the POSSUM library routine PRODIR (see the P/OS System Reference Manual) to create and delete directories.

6.1.3 File Names

A file name is a one- to nine-character alphanumeric string that is generally used as a mnemonic name to identify a particular file within a directory. Some valid file names are:

P/OS FILE SPECIFICATIONS

ACCOUNTS

001005

INDEX3

MAIL

Notes

- In a file specification, use a period (.) to separate the file name field from the file type field.
- If a file specification contains no file name field, P/OS does not supply one by default.

6.1.4 File Types

A file type is a three-character alphanumeric string that is generally used to categorize a file. P/OS uses a set of standard file types to provide useful defaults. For example, the file type TSK indicates that the file is an executable program (task image).

Table 6-2: User-visible File Types

<u>File</u>	<u>Type</u>
BASIC program	BAS
data file	DAT
document file	DOC
text file	TXT

Table 6-3: System/Application File Types

<u>File</u>	<u>Type</u>
forms library	FLB
system file	SYS
task image	TSK
converted help file	HLP
converted message file	MSG
converted menu file	MNU

In a P/OS file specification, use a semicolon (;) to separate the file type field from the version number field.

6.1.5 Version Numbers

A version number is a number that uniquely identifies files that have the same file name and file type. On VAX/VMS and P/OS, version numbers are decimal. On RSX-11M/M-PLUS systems, version numbers are octal.

P/OS FILE SPECIFICATIONS

When you create a file that does not already exist, it is assigned version number one by default. When you create a file that already exists, it is assigned the next highest version number by default.

6.1.6 Wild Card Conventions

A full set of wild card conventions are available from PRO/Tool Kit DCL. If you are using the PRO/Tool Kit, see the Command Language/Utilities Manual for more information. If you are using the host Tool Kit, refer to your host system command language manual.

6.2 LOGICAL NAMES

A logical name is a user- or system-defined name for all or part of a file specification. You can substitute a logical name for a device, directory, or file name either interactively or from within a program.

Programs that use logical names can be independent of specific devices, directories, and files. At run-time, your program can make connections between its logical names and actual physical names.

A logical name always refers to an associated equivalence name. The system provides a logical name facility that translates a logical name and returns its equivalence name. An equivalence name can be a physical device, directory, or file specification, or can itself be a logical name. PRO/RMS-11, however, accepts physical device names, volume labels, or other logical names as equivalence names (see Section 6.2.2). Any number of logical names can have the same equivalence name.

The system stores logical name strings and their equivalence strings in a single logical name table that cooperating tasks can use. The system uses this table when translating logical names to equivalence names.

Within the strict context of the logical name facility, the logical name and its equivalence name are simply character (byte) strings. The only restrictions to logical name strings and equivalence name strings are:

- The string length must not exceed 255(10) bytes.
- There must be an equivalence name string for each logical name string entered in the logical name table.

LOGICAL NAMES

6.2.1 System Logical Names

Several logical names are defined and used by P/OS. Table 6-4 shows the P/OS system logical device names. Table 6-5 shows the P/OS system logical directory names.

NOTE

Tasks can refer to P/OS logical names but must not reassign them. P/OS may run unpredictably or stop processing if those names are reassigned.

Table 6-4: P/OS System Logical Device Names

<u>Device</u>	<u>Logical Name</u>	<u>Equivalence Name</u>
RD50/51 Hard Disk	BIGDISK:	_DW001:
	BIGVOLUME:	_DW001:
	DW001:	BIGVOLUME:
	LDW001:	BIGDISK:
	SY000:	(current default device)
	SYSDISK:	LB000:
RX50 diskette	DISKETTE1:	_DZ001:
	DISKETTE2:	_DZ002:
	LDZ001:	DISKETTE1:
	LDZ002:	DISKETTE2:
	USERDISK:	SYSDISK: *
	DZ001:	**
	**	_DZ001:
	DZ002:	**
	**	_DZ002:

* The data diskette (P/OS Diskette)

** Assigned at volume mount time (see Section 3.2.3)

LOGICAL NAMES

Table 6-5: Logical and Equivalence Directory Names

<u>Logical Name</u>	<u>Directory Name (P/OS Hard Disk)</u>	<u>Directory Name (P/OS Diskette)</u>
APPL\$DIR	SYSDISK:[ZZAPnnnnn]	SYSDISK:[ZZAPPL]
APPL\$MENU	SYSDISK:[ZZAPnnnnn]x.MNU	SYSDISK:[ZZAPPL]x.MNU
APPL\$HELP	SYSDISK:[ZZAPnnnnn]x.HLP	SYSDISK:[ZZAPPL]x.HLP

6.2.2 PRO/RMS-11 Translation of Logical Names

As part of I/O processing in program execution, PRO/RMS-11 translates logical names and returns their equivalence names. The following conventions govern PRO/RMS-11 translation of logical names:

1. PRO/RMS-11 translates only those logical names occurring within the context of a valid device specification. PRO/RMS-11 does not translate logical names that refer to directories or files.
2. PRO/RMS-11 continues to translate logical name strings until it:
 - encounters an equivalence name string beginning with an underscore () character.
 - fails to translate a string.
 - encounters an equivalence name string not ending with a colon (:).
 - reaches the maximum number of translations allowed (eight).

6.2.3 FILES-11 ACP Use of Logical Names

The Files-11 Ancillary Control Processor (ACP) creates two logical names when it mounts a file-structured disk or diskette:

1. The volume label specified at the time the volume was initialized is equated to the physical device on which the volume is mounted.
2. The physical device name is equated to the volume label.

For example, if you mount a diskette volume named "FINANCE" in drive 1, the ACP would assign:

LOGICAL NAMES

<u>Logical Name</u>	<u>Equivalence Name</u>
FINANCE:	_DZ001:
DZ001:	FINANCE:

An application program can reference the disk with the volume label FINANCE by using the logical name FINANCE:. PRO/RMS-11 translates the logical name to determine the actual physical device. Similarly, the application can determine the volume that is currently mounted in a specific drive.

6.2.4 Manipulating Logical Names

PRO/Tool Kit DCL provides several commands for manipulating logical names: ASSIGN, SHOW LOGICAL, and so forth. See the PRO/Tool Kit Command Language/Utilities Manual for more information.

A program can call the POSSUM library routine PROLOG (see the P/OS System Reference Manual) to perform the following logical name functions:

- Create a logical name for a device specification.
- Delete a logical name for a device specification.
- Translate a logical name to a device specification.

Similarly, a program can issue the CLOG\$, TLOG\$, and DLOG\$ directives (see the P/OS System Reference Manual) to perform the following logical name functions:

- Create a logical name string (CLOG\$)
- Delete a logical name string (DLOG\$)
- Translate a logical name string (TLOG\$)

6.3 ACCESSING APPLICATION FILES

Most applications consist of several files: task images, data files, menu files, help files, and so forth. Unless explicitly placed elsewhere, these files exist in the application directory.

On a P/OS Diskette system, there is only one application directory, [ZZAPPL]. Thus, there is no problem locating the files.

ACCESSING APPLICATION FILES

On a P/OS Hard Disk system, however, application directories created by Disk/diskette Services have the form [ZZAPnnnnn], where nnnnn is an integer value that is entirely site-dependent.

In order to access its files, a task image passes file specifications to PRO/RMS-11 (usually via a language run-time system) that contain the name of the application directory. That name is available in the form of a logical name, APPL\$DIR, assigned by P/OS at run-time.

PRO/RMS-11, however, cannot translate logical directory names (see Section 6.2.2). Thus, a task must translate APPL\$DIR in order to determine the physical name of its application directory. The methods for translating logical names are described in Section 6.2.4.

SYSDISK is the logical name of the device (disk or diskette) that contains the application directory. Thus, on a Hard Disk system, APPL\$DIR generally translates to:

SYSDISK:[ZZAPnnnnn]

You can postpone the translation responsibility during program development by using Fast Install, rather than Disk/diskette Services, to install your application. With Fast Install, you create the application directory yourself and can use its name as a constant in your application.

6.3.1 Menu and Help Files

The installation command (.INS) file provides an easy way for an application to access a menu file and a help file. If you provide an ASSIGN MENU line and an ASSIGN HELP line in your .INS file, the system will automatically open the specified file whenever you call a User Interface Library (POSRES) routine that uses it. If your program uses additional menu and/or help files, it must explicitly open them.

6.3.2 Running Applications from the PRO/Tool Kit

The PRO/Tool Kit is an installed application. Thus, while the PRO/Tool Kit is running, the logical name APPL\$DIR equates to the directory that contains the PRO/Tool Kit files. If your application uses external files and you want to run it from PRO/Tool Kit DCL, use one of the following techniques:

1. When opening files, specify the physical name of the application directory. Do not translate APPL\$DIR. This is only a temporary state. You will have to modify your application to translate APPL\$DIR before distributing it.

ACCESSING APPLICATION FILES

2. Place your application files in the PRO/Tool Kit directory so that translating APPL\$DIR will work correctly. Be sure to remove the files when you have completed your application.

NOTE

Do not reassign APPL\$DIR. If you do, the PRO/Tool Kit will not work correctly.

It is recommended that you use the first technique. In general, the PRO/Tool Kit application directory should be left unchanged.

CHAPTER 7

TASK BUILDING

The Professional Application Builder (PAB) is a utility that links your object modules (.OBJ files) with system software, producing task image files that can be executed on P/OS. You control PAB by means of two command files that are explained in detail later on.

PAB is a somewhat cryptic but extremely powerful tool. This chapter provides enough practical information about PAB to task build simple applications. Your documentation set includes the RSX-11M/M-PLUS Task Builder Manual, which describes PAB in great detail. If you are developing large or sophisticated applications, it is recommended that you read the manual and become proficient in the PAB languages. PAB has powerful features that allow you to optimize task images in many different ways.

7.1 INVOKING PAB ON THE PRO/TOOL KIT

If you are using the PRO/Tool Kit, invoke PAB with the LINK command which is described in detail in the Command Language/Utilities Manual.

```
$ LINK @file
```

The default file type is ".CMD". Alternatively, you can type:

```
$ LINK  
File(s)? @file
```

7.2 INVOKING PAB ON VAX/VMS

On VAX/VMS, the PAB executable image is called PROTKB.EXE. It runs under the AME (Application Migration Executive) and supports named as well as numbered directories. Insert the following symbol in your LOGIN.COM file:

INVOKING PAB ON VAX/VMS

```
$ PAB := $PROTKB
```

Once the symbol is defined, you can use a single command to invoke PAB:

```
$ PAB @file
```

The default file type is ".CMD". Alternatively, you can type:

```
$ PAB  
PAB> @file
```

7.3 INVOKING PAB ON RSX-11M/M-PLUS (DCL)

On RSX-11M/M-PLUS, the PAB executable image is called PROTKB.TSK. If PAB is installed on your system as "...PAB", you can invoke it with the following command:

```
$ PAB @file
```

If PAB is not an installed task on your system, invoke it with the following command:

```
$ RUN $PROTKB  
PAB> @file
```

The same commands also work for MCR.

7.4 BUILDING APPLICATIONS

As mentioned in Chapter 4, the implementation phase requires that you create two PAB files (a .CMD file and an .ODL file) for each task image in your application. Each file contains commands in a different language that tell PAB exactly how to build the desired task image.

Some of the Tool Kit languages include software for creating PAB files that automatically contain language-specific information as well as information for PRO/RMS-11. Others simply describe the required files and expect you to create them yourself with an editor. In either case, create the required files and examine their contents and format.

Because no language can anticipate all of your task's requirements, you will almost certainly have to make some edits to those files before task building. The following tools require specific information in your PAB files:

BUILDING APPLICATIONS

- CORE Graphics Library
- PRO/FMS-11
- PRO/RMS-11
- POSRES User Interface Library
- POSSUM System Services

If your task uses any of those tools, turn to the documentation for each tool and make the appropriate edits.

NOTE

Most of the high-level languages use POSRES User Interface Library routines (particularly RDMSG and WTRES) for run-time support and thus require some POSRES support in your .CMD file. POSRES is described in detail in Chapter 8.

7.5 THE COMMAND (.CMD) FILE

The PAB command (.CMD) file specifies input and output files and contains option lines that specify cluster libraries, buffer sizes, logical unit number assignments, and so forth.

Figure 7-1 shows a sample .CMD file, written for a MACRO program that uses POSSUM and PRO/RMS-11.

```
SAMPLE=SAMPLE/MP
CLSTR=POSSUM,RMSRES:RO
//
```

Figure 7-1: Sample PAB Command File

The double-slash (//) indicates the end of the file. The rest of the file is explained in the following sections.

7.5.1 The Command Line

The first line is a command that specifies input and output files and switches. Input files go on the right side of the equal sign and output files go on the left. For example:

```
SAMPLE=SAMPLE/MP
```

THE COMMAND (.CMD) FILE

This command specifies one input file and one output file, both named SAMPLE. The /MP switch specifies that the input file is an overlay descriptor language (.ODL) file as described in Section 7.6. The output file is a task image (default file type .TSK).

The use of an .ODL file does not imply that your task must be overlaid to run on P/OS. It simply describes your task image in more detail than is possible in the .CMD file.

NOTE

If an existing PAB command file has a /-CP switch on the output file, remove it. Use /CP (the current default) for all tasks built for P/OS.

If an existing PAB command file has a /-FP switch on the output file, remove it unless no floating point instructions are used by your task. Use /FP (the default) for all tasks built for P/OS V1.7 or later. P/OS provides EIS libraries for upwards compatibility purposes only.

If you would like a map that shows exactly how your task loads into memory, specify a second output file, such as:

```
SAMPLE,SAMPLE/MA/-SP=SAMPLE/MP
```

This command specifies a task image and a load map (.MAP) file. The /MA switch specifies that the load map is to contain the names of the system library routines used in your task. The /-SP switch prohibits automatic printing of the load map on host Tool Kit systems.

7.5.2 The CLSTR Option

The CLSTR option specifies the cluster libraries used by your task. If your task references a non-null-rooted cluster library, it must be the first (default) library in the CLSTR option. This applies to high-level language run-time libraries, such as PASRES and PBFSML. If all are null-rooted, the first library called by your task becomes the default. For example, if a PASCAL program uses PRO/RMS-11 and POSRES services, the cluster option would be:

```
CLSTR=PASRES,POSRES,RMSRES:RO
```

THE COMMAND (.CMD) FILE

NOTE

Do not embed any spaces in the CLSTR option. Also, do not include a comment on the CLSTR option. It will cause PAB to return a fatal option syntax error.

The ":RO" switch specifies read-only access. For P/OS V1.7, the overlay run-time system was modified to allow read/write access to non-default (in addition to default) clustered libraries that have not been installed read-only. Such libraries may be useful for:

- passing information between cooperating application tasks (the tasks should provide their own access synchronization).
- extending the effective available read/write virtual memory usable for task impure data.

In both cases, the task(s) must ensure that the library is mapped by calling a routine in the library that does not return to the caller until any access to the read/write data is completed. This is not normally necessary for a non-null-rooted default cluster member, since it is usually already mapped as desired.

7.5.3 NULLIB

The special non-null-rooted default cluster member NULLIB is provided for two purposes:

- It can guard against potential memory fragmentation problems that might cause task deadlock in certain instances.
- It can provide better performance in cases when a null-rooted cluster member would otherwise become the 'effective' default member of the cluster and would be unnecessarily re-mapped (and potentially re-loaded from disk). Assuming that the application would access cluster members other than the first one accessed, re-mapping that first member after every access to some other one could prove costly.

However, the increased physical memory now standard for P/OS makes memory fragmentation problems considerably less likely for most applications. Also, now that the RMSRES and POSSUM resident libraries are fixed in memory, there is no possibility of disk loading overhead when one of these becomes the effective default member of a cluster.

As a general rule, when all members of a library cluster have null roots, your application should attempt to ensure that the first library accessed (which will become the effective default cluster member) is the one that the application will refer to most frequently.

THE COMMAND (.CMD) FILE

This will minimize the likelihood of unnecessary mapping and possible disk loading.

7.6 THE OVERLAY DESCRIPTOR LANGUAGE FILE

The overlay descriptor language (.ODL) file specifies the object modules used by your task (some of which are automatically extracted from libraries) and describes how your task image will use its 32K words of virtual memory.

Each high-level language and development tool specifies which object modules and/or libraries to include in your .ODL file, along with your own object modules.

NOTE

Some tools may say to use files in LB:[1,1], the RSX system library directory. The host Tool Kit system library directory is LB:[1,5], not LB:[1,1]. Thus PAB automatically replaces all references (including defaults) to LB:[1,1] with LB:[1,5].

Normally, PAB allocates memory for each object module in linear fashion (while automatically extracting object modules from SYSLIB.OLB as needed). In order to reduce the size of a task, you can create a structure where one or more modules are overlaid (share the same memory). That subject is discussed in detail in the RSX-11M/M-PLUS Task Builder Manual.

The following is an example of an .ODL file, written for a MACRO-11 program that uses PRO/RMS-11:

```
.ROOT    USER-RMSROT
USER:    .FCTR    FIRST-USERSUB
@LB:[1,5]RMSRLX
.END
```

The first line is a .ROOT directive, which defines the structure and contents of the task image. Although it can contain object modules, .ROOT usually contains references to symbols defined elsewhere in the .ODL file, as is the case here: USER and RMSROT are symbols.

The second line is a .FCTR directive, which defines the symbol USER as two object modules: FIRST and USERSUB. To specify additional object modules, you could add them to this .FCTR directive or include additional .FCTR directives as needed.

THE OVERLAY DESCRIPTOR LANGUAGE FILE

The hyphen specifies that FIRST and USERSUB are to be concatenated (each is to have its own area of memory). A comma would specify that the modules are to be overlaid. In that case, PAB would allocate a single area of memory, usable by both modules, but only one at a time.

The third line includes the PRO/RMS-11 overlay descriptor language file RMSRLX.ODL, which defines the symbol RMSROT. The at-sign character (@) allows you to nest .ODL files in the same way that you can nest program source code.

CHAPTER 8

P/OS USER INTERFACE SERVICES

The P/OS user interface services provide the means by which users* can interact with your application in exactly the same manner that they interact with P/OS. From the user's point of view, the transition from P/OS to application can be just another menu.

The user interface services also allow you to remove all text from your source code, making a single application usable in any number of languages. For example, you can package the same task images with menu, help, and message files in English, French, German, Italian, and so forth.

This chapter describes how to design and implement a menu-based user interface. To accomplish that, you must be familiar with how a menu interface works from a user's point of view. This chapter assumes that you are familiar with the P/OS user interface. If not, it is recommended that you spend some time working with the P/OS menus and help structure before proceeding.

8.1 OVERVIEW

A menu-based user interface allows the user to interact with your application by repeatedly selecting one or more options** from a finite list or "menu." Because all options are visible to the user, there is no necessity to memorize a command language.

In practice, the structure that best matches the user's perception of a menu interface is the multi-way tree (hierarchy) where each option on a menu represents a decision and "points" to another menu, and so

* In this chapter, the term "user" means the person who actually acquires, installs, and uses your application.

** P/OS end-user documentation uses the term "item" rather than "option."

OVERVIEW

forth. The bottom or "leaf nodes" of the tree represent states in which your application has gathered enough information to perform an operation. When the operation is complete, the user repeats the decision making process beginning at the "main menu" or at some other menu in the tree.

Because the menu interface is entirely under program control, the user need not be restricted to downward movement in the menu tree. You can allow instantaneous movement, forward or backward, to any other menu in the tree. You can use backward movement to provide the user with a way to cancel a wrong decision and start over, and forward movement to provide sophisticated users with an abbreviated or concise way to make decisions.

While a menu is active, some function keys are "trapped" and used as part of the option selection process (described in detail later on.) Other function keys return control to your application with the identity of the key that was pressed. This chapter contains some recommendations on how to assign semantic meanings to and process those keys.

A menu-based user interface also includes context-sensitive help in the form of help menus, which are similar to control menus, and help text frames, which are simply informative displays. Help structures can be multi-way trees or complex networks of interlocking menus and text frames.

You can associate help structures with menus or with individual options on menus. While a menu is active, the menu interface automatically activates the appropriate help structure whenever the user presses the HELP key. If no menu is active, your application can detect and process requests for help by explicitly activating a help structure. You can even monitor the user's progress through the menu tree and offer help if the user makes repeated errors or seems to be stalled at some point.

A menu-based user interface also provides a way for your application to display context-sensitive messages that announce error conditions, confirm completion of operations, and so forth. Each time it displays a menu, your application can specify up to two lines of messages to appear with the menu at a predefined location. Your application can also send a message to the "View Message/Status" service on the P/OS Main Menu.

OVERVIEW

FORMS

Profile for Single Choice Menu MAIN

Frame Description [This is the main menu for the elementary education application.]
Global Help Frame (MENU) Default Option (3) Global Action String
[DATABASE = ELEM]

Display for Single Choice Menu MAIN

[ELEMENTARY EDUCATION APPLICATION]	
[This application offers elementary education in seven fields of study. Select one of the courses listed here.]	
[BIOLOGY COMPUTER SCIENCE GEOGRAPHY GOVERNMENT HISTORY LITERATURE MATHEMATICS
[Make a selection and press the DO key:]	

Action Number 3 for Single Choice Menu MAIN

Description: GEOGRAPHY	
Action Description [This is the description of the Geography option.]	
Option Keyword [GEO]
Option Help Frame [HELPGEO]	
[GEOG0010	Option Action String

FDT

MENU DEFINITION FILE

FRAME 1 "MAIN"	FRAME 2	FRAME 3	FRAME 4	FRAME 5	FRAME 6	FRAME 7	FRAME 8	FRAME 9	FRAME 10	FRAME 11
-------------------	---------	---------	---------	---------	---------	---------	---------	---------	----------	----------

POSRES

MENU

ELEMENTARY EDUCATION APPLICATION	
This application offers elementary education in seven fields of study. Select one of the courses listed here:	
	BIOLOGY COMPUTER SCIENCE -> GEOGRAPHY GOVERNMENT HISTORY LITERATURE MATHEMATICS
Additional Options available	
Make a selection and press the DO key:	

Figure 8-1: The User Interface Tools

OVERVIEW

The user interface services consist of:

- **The Frame Development Tool**

The Frame Development Tool (FDT) is a utility program (you may prefer to think of it as an editor) that allows you to create menu, help, and message frames, to store frames in files, and to convert the files into executable format. FDT is described in detail in the Tool Kit Reference Manual.

- **The POSRES User Interface Library**

POSRES is a P/OS cluster library containing a set of callable routines that manage menus and help structures (including frames stored in files) and process function keys. This chapter provides a general description of how to use the POSRES routines. A detailed description of each routine can be found in the Tool Kit Reference Manual.

Figure 8-1 shows the relationship between FDT and POSRES. FDT allows you to create a frame (in this case, a single-choice menu) by filling in forms. The frame description is then stored in a file and installed in the application directory along with the rest of your application files. At run-time, your application uses POSRES routines to open the frame file, retrieve the frame, and activate it.

8.2 DESIGNING A MENU STRUCTURE

The primary goals in designing a good menu tree are consistency and friendliness. The user should never feel lost or trapped. You should always provide a way to back out of a wrong decision.

Convenience is also very important. On any menu, include options that are related by function (rather than logic) so that common operations can be completed with minimal switching of menus. If necessary, put the same option on more than one menu. For example, P/OS File Services and Disk/Diskette Services both allow you to show your current directory, show the contents of a directory, and so forth.

8.2.1 Format of a Menu

From the user's point of view, there are three types of menus: single-choice menus, multiple-choice menus, and help menus. They all have roughly the same fields. Figure 8-2 shows a single-choice menu with the fields pointed out.

DESIGNING A MENU STRUCTURE

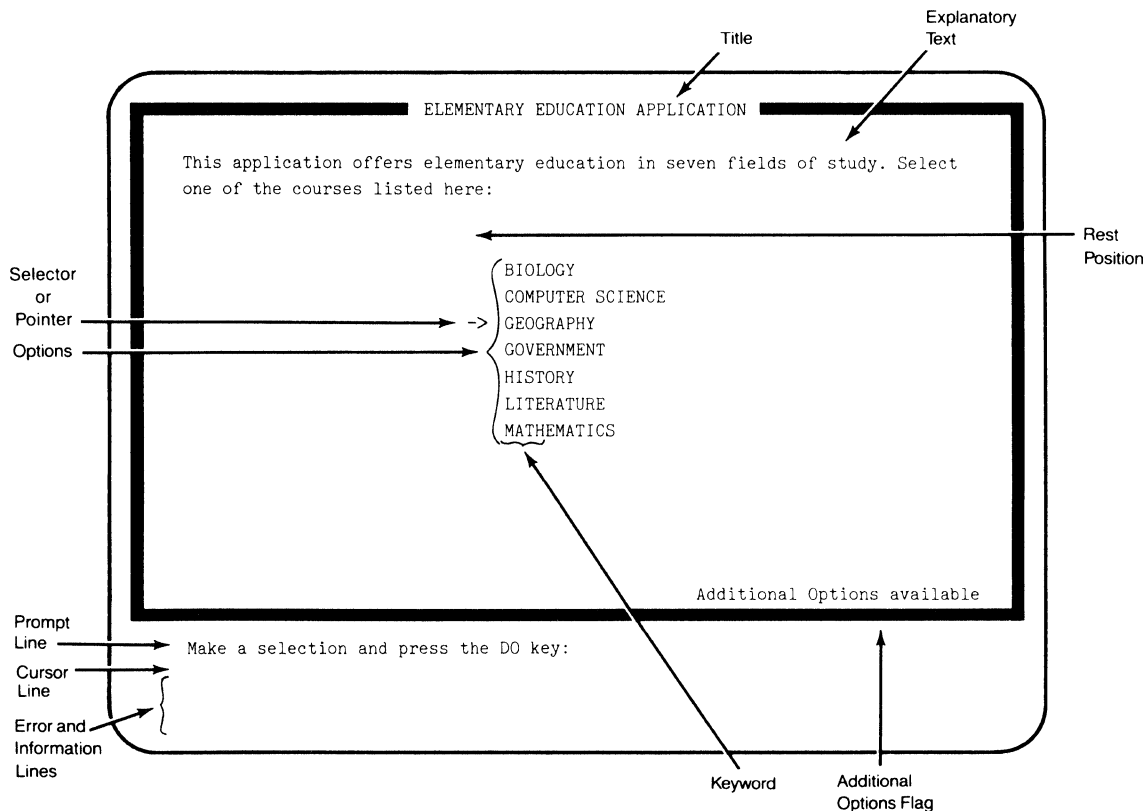


Figure 8-2: Single-choice Menu

On a single-choice menu, part of each option appears in boldface in order to indicate which characters the user can type to make an unambiguous choice. The boldface characters are called the "keyword" for the option.

8.2.2 Single-Choice Menus

There are actually two kinds of single-choice menu, static and dynamic, which is explained later on. The difference is relevant only to programmers; from the user's point of view, there is only one.

The user selects one from a list of up to 12 options. There are two ways to select an option:

- The user positions the selector at the desired option by pressing the Up Arrow and Down Arrow keys, then presses the DO key. When the menu is displayed, the selector begins at the default option or, if none was specified, at the rest position.

DESIGNING A MENU STRUCTURE

- The user positions the selector at the desired option by typing keyword characters, then presses the DO key. The selector moves only when the characters identify a unique choice.

Single-choice menus are normally used for verbs (program control). For example, P/OS File Services has a single-choice menu with options like "Copy file," "Delete file," "Display file," and so forth.

```
COURSE: GEOGRAPHY OF NORTH AMERICA

This course covers rivers, lakes, and mountains. The first topic is mountains.
The mountains listed below are located in different parts of the world. Select
all the mountains located in North America:

->
    MT. RUSHMORE
    MT. WASHINGTON
    MT. HOOD
    GRAND TETON
    MT. RAINIER
    MT. FUJI
    KILIMANJARO
    MT. ST. HELENS
    MT. MCKINLEY

Choose one or more options with the SELECT key and press the DO key:
```

Figure 8-3: Multiple-choice Menu

8.2.3 Multiple-Choice Menus

On a multiple-choice menu, the user chooses one or more options from a list. The list can be any length, possibly covering several screens. Figure 8-3 shows a multiple-choice menu.

The user selects options by repeatedly positioning the selector (with the up arrow and down arrow keys) and pressing the SELECT key. A non-blinking arrow remains next to each selected option. The user can deselect an option by pressing the SELECT key again, or can deselect all options (start over) by pressing the CANCEL key.

DESIGNING A MENU STRUCTURE

If the option list covers more than one screen, the user can move forward or backward in the option list by pressing the PREV SCREEN and NEXT SCREEN keys. The DO key signals that the selection process is complete.

Multiple-choice menus are normally used for objects (data), rather than commands. A common example is a list of files or directories.

8.2.4 Key Processing in Menus

This section describes key processing in menu option selection. Appendix D lists all of the key codes and labels.

- Keyboard keys are accepted if they match an option keyword. Otherwise, the keyboard bell rings and selection continues. The delete key deletes the previously typed character.
- The ADDTNL OPTIONS key returns control to your application if your menu display call specified Additional Options. Otherwise, the keyboard bell rings and option selection continues.
- The Up Arrow and Down Arrow keys move the selector up and down. If the user tries to move the selector out of range, the keyboard bell rings and selection continues.
- The CANCEL key moves the selector to the rest position. On a multiple-choice menu, it deselects all options.
- The DO key selects an option on a menu and returns control to your application. On a multiple-choice menu, it selects the current option and returns all other selected options. If no option has been selected (the pointer is at the rest position) it rings the keyboard bell and continues.
- The HELP key activates a help structure as described in Section 8.5.3.
- The HOLD SCREEN key functions normally.
- The INTERRUPT key does not by itself return control to your application. If followed by the DO key, the system aborts your application and displays the P/OS Main Menu. Otherwise, option selection continues as usual.
- The PRINT SCREEN key functions normally.

DESIGNING A MENU STRUCTURE

- The SELECT key selects or deselects an option on a multiple-choice menu. It does not return control to your application. The DO key selects an option and returns control with all selected options.
- Other function keys either ring the keyboard bell or terminate the menu display and return control to the executing task with a numeric code to identify which key was pressed. In general, you should handle invalid keystrokes by redisplaying the same menu with a message something like "That key is invalid here - please try again."

8.2.4.1 Action Strings

Action strings are character strings that are stored in a menu definition but not displayed; POSRES returns them to your program with specific calls. Their purpose is to associate menus and options with data usable by your task. For example, you can associate menu and options with other menus, tasks, subroutines, or callable services.

There are two types of action strings:

- **Global action strings**

A global action string associates data with a single-choice menu. Your application can obtain and use the global action string each time it reads a new menu from the menu file.

- **Option action strings**

Option action strings associate data with options on a single-choice menu. A successful option selection returns an option action string to your application.

8.2.4.2 Option Keywords

On a single-choice menu, part of each option must be designated the keyword. The user can select the option by typing keyword characters, rather than by pressing arrow keys.

The keyword must be at least one character. It can be any contiguous substring of an option or the entire option. For example, you could use any of the following:

Enter Accounts Payable

Enter Accounts Payable

DESIGNING A MENU STRUCTURE

Enter Accounts Payable

A keyword must be unique within the menu. No keyword can be a substring of another keyword. For example, the string "1" is a substring of the string "10".

Verbs are usually the most appropriate keyword. If no keyword is suitable, use numbers (this is not recommended). For example:

- 1 Sales Report: Area One
- 2 Sales Report: Area Two
- 3 Sales Report: Area Three

8.3 IMPLEMENTING A MENU STRUCTURE

From a developer's point of view, there are two kinds of menus:

- **Static menus**

Static menus are created with FDT and stored in library files. In general, static menus are relatively easy to program and are intended for program control.

- **Dynamic menus**

Dynamic menus are created by your application at run-time. In general, dynamic menus are intended for manipulating data.

NOTE

Only single-choice menus can be static or dynamic.
Multiple-choice menus are always dynamic.

POSRES uses three buffers for temporary storage of menu frames, one each for static single-choice menus, dynamic single-choice menus, and multiple-choice menus. You allocate memory for these buffers when you build your program (see Section 8.8). Because there is only one buffer of each type, you can have only one menu of each type in memory at one time.

8.3.1 Displaying Menus

All of the routines for displaying menus accept a parameter that specifies whether to display the Additional Options message and return control if the user presses the ADDTNL OPTIONS key. P/OS generally uses Additional Options menus for services beyond those offered on the current menu. For example, the OLDFIL file selection menu uses

IMPLEMENTING A MENU STRUCTURE

Additional Options to allow the user to type a file specification, rather than select files from a list.

8.3.1.1 Static Single-Choice Menus

Static single-choice menus are created with FDT and stored in menu definition files. There are two ways to open a menu definition file:

- The ASSIGN MENU command in your installation command file (see the Tool Kit Reference Manual) opens a menu file at run-time. If your application uses only one menu file, that's all you have to do.
- The Open Menu File (MFILE) routine explicitly opens menu files. To use it, however, you must specify the name of the directory that contains the menu file. For a P/OS Hard Disk application, that requires translating the logical name APPL\$DIR (see Section 6.3).

To display static-single choice menus, use the following POSRES routines:

- **Read Menu Frame (MFRAME)**

MFRAME reads a specified menu from the menu file into the static buffer.

- **Display Single-Choice Menu (MENU)**

MENU displays the menu in the static buffer and processes user keystrokes (as described in Section 8.2.4). You can specify up to two message lines to appear on the menu.

8.3.1.2 Dynamic Menus

Dynamic menus are created by your application at run-time. To create a dynamic menu, you must first clear the appropriate buffer (dynamic single-choice or multiple-choice) and then pack it. The POSRES menu packing routines are:

- **Pack Dynamic Single-Choice Menu (DPACK)**

DPACK packs the dynamic buffer with a new menu.

- **Pack Multiple-Choice Menu (MPACK)**

IMPLEMENTING A MENU STRUCTURE

MPACK packs the multi-buffer with a new menu. It includes parameters for setting the maximum number of options the user can select and for receiving the responses.

Both routines accept a parameter that clears the buffer. They also accept any number of field parameter groups, each of which specifies the contents of a menu field (see Figure 8-2). Thus, you can pack a menu buffer with a single routine call or several calls.

You also can use the contents of the static buffer to create a dynamic menu. The POSRES menu unpacking routine is:

- **Unpack Menu Buffer (MUNPK)**

MUNPK unpacks the menu in the static buffer so that its contents can be modified and reused as a dynamic single-choice or multiple-choice menu.

Once you have packed the dynamic buffer, display the menu by calling one of the display routines:

- **Display Dynamic Menu (DMENU)**
- **Display Multiple-Choice Menu (MMENU)**

These routines display the menu in the appropriate buffer and process user keystrokes (as described in Section 8.2.4). You can specify up to two message lines to appear on the menu.

8.3.2 Programming with Menus

When control returns to your application from a menu display routine, the menu remains visible on the screen. Examine the first word of the status block to see what happened.

If it contains +1, option selection was successful and the second word of the status block contains the ordinal number of the selected option. You can branch on that value or use the option action string (if defined).

If the first status word contains -14, the user pressed a function key other than DO and the second word contains one of the function key codes shown in Appendix D. If your menu display call specified Additional Options, be sure to check for 14 in the second status block word.

IMPLEMENTING A MENU STRUCTURE

Your application must determine which function keys are valid and which are not in the context of the current menu. Section 8.2.4 provides some suggestions on how to process function keys. If the user pressed an invalid key, redisplay the same menu with a helpful message. A program designed for novice users could count the number of consecutive invalid keys and offer help if and when appropriate.

Any other value indicates that an error has occurred. Appendix C shows all of the possible values. While debugging, you may find it helpful to dump the status block whenever an error occurs. Although no application can anticipate all errors, try to anticipate and handle those that a user might encounter.

When processing is complete, use the Close Menu File (MCLOSE) routine to close the menu definition file (if open) and exit. Otherwise, the file remains open until your task exits.

8.3.3 File Specification Routines

POSRES provides two special user interface routines for working with files.

8.3.3.1 New File Name (NEWFIL)

The New File Name routine activates the P/OS New File Specification form shown in Figure 8-4 and returns a full file specification in the form:

```
dev:[directory]filename.typ;version
```

The user can fill in the form and press DO, or can press ADDTNL OPTIONS. POSRES automatically displays an Additional Options menu that allows the user to: specify a new file type, specify a different directory, or enter an extended file specification.

When control returns to your application, reset menu and help contexts. If your installation command file contains ASSIGN MENU and/or ASSIGN HELP commands, the next POSRES call will automatically reopen those files. If not:

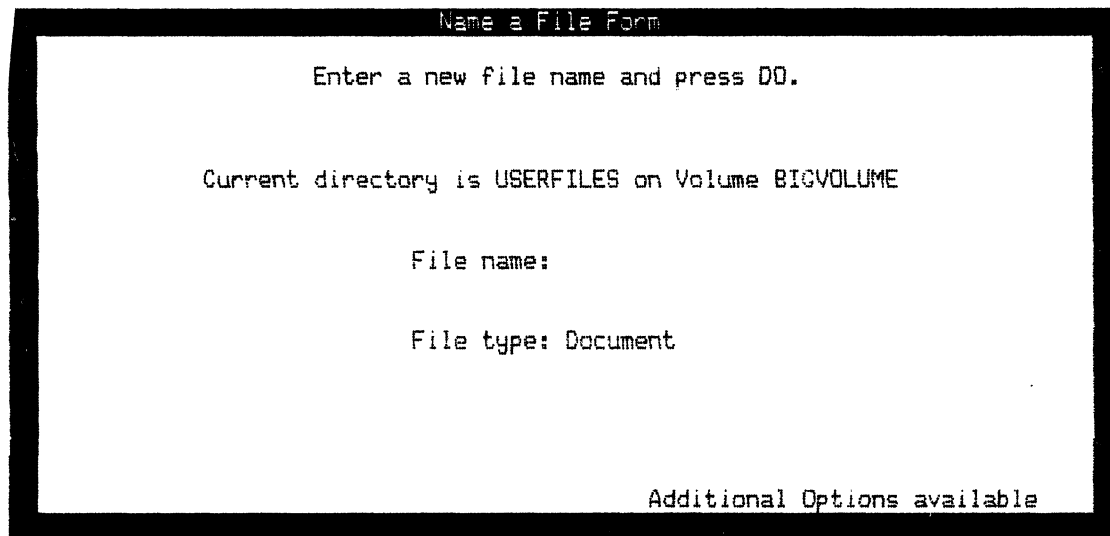
- open the menu file explicitly with MFILE
- reset the menu frame with MFRAME

IMPLEMENTING A MENU STRUCTURE

- open the help file explicitly with HFILE
- reset the help frame with HFRAME

8.3.3.2 Old File Name (OLDFIL)

The Old File Name routine activates the P/OS File Selection Menu and returns the full specifications of one or more selected files. You can supply a wild-card string to specify a subset of the files in the user's current directory and can specify the maximum number of selections. For example, you can use the default wild-card specification (*.*) by supplying a zero-length string. That displays the latest versions of all files in the user's current directory. (Refer to the Command Language/Utilities Manual or your host system documentation for wild-card syntax.)



The image shows a screenshot of a terminal window titled "Name a File Form". The text inside the window is as follows:

```
Name a File Form

Enter a new file name and press DD.

Current directory is USERFILES on Volume BIOVOLUME

File name:

File type: Document

Additional Options available
```

Figure 8-4: "Name a File Form"

An Additional Options menu allows the user to choose a different directory or volume, enter an extended file name, show all versions of the files, or show only the latest versions of files. The last two options, however, work only when you use the default wild-card specification or when you pass a wild-card specification that specifies an asterisk (*) for the version number. Otherwise, when the user selects "Show all versions" or "Show latest version", the same file selection menu will redisplay.

IMPLEMENTING A MENU STRUCTURE

When control returns to your application, reset menu and help contexts. If your installation command file contains ASSIGN MENU and/or ASSIGN HELP commands, the next POSRES call will automatically reopen those files. If not:

- open the menu file explicitly with MFILE
- reset the menu frame with MFRAME
- open the help file explicitly with HFILE
- reset the help frame with HFRAME

8.4 DESIGNING A HELP STRUCTURE

The primary goal in designing a good help structure is to guide the user smoothly through the menu tree. Although it's important to anticipate where a user might become unsure how to proceed, there should be no point in the menu tree where the HELP key produces nothing but a beep. The help structure is an area where technical writers often produce the best results.

Help structures consist of help menus and help frames. Help menus do not provide information; they simply guide the user to the appropriate help text frame or another help menu. Help text frames provide information about your application. The quality of this on-line documentation can do much to enhance its market image.

The help frames at the "root" of the menu tree should contain general information. The frame at the "leaf nodes" should contain more specific information. Although you can assume that the user is familiar enough with the menu system to select an option from a menu, it is recommended that you always state which keys are expected, for example: "Make a selection and press DO.", "Press NEXT SCREEN to continue.", and so forth.

8.4.1 Help Menus

On a help menu, the end user chooses one from a list of up to 12 options. Figure 8-5 shows a Help menu. The format and option selection process are similar to a single-choice menu.

DESIGNING A HELP STRUCTURE

8.4.2 Key Processing in Help Menus

This section describes key processing in help menus. Appendix D lists all of the key codes and labels.

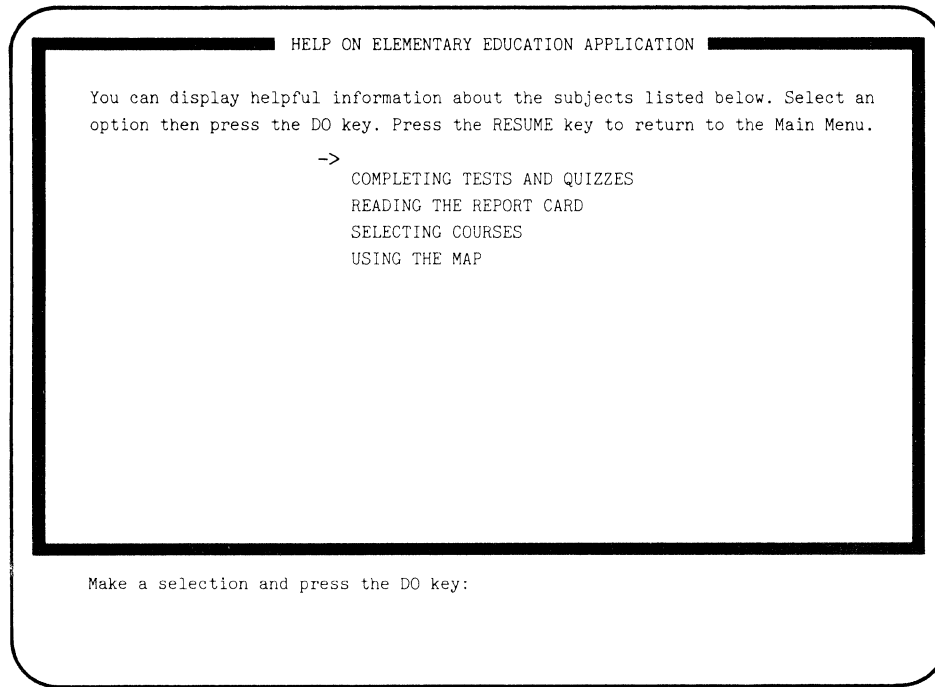


Figure 8-5: Help Menu

On a help menu, option selection keys are processed as follows:

- Main keyboard keys are accepted if they match an option keyword. Otherwise, the keyboard bell rings and selection continues. The delete key deletes the previously-typed character.
- The Up Arrow and Down Arrow keys move the selector up and down. If the user tries to move the selector out of range, the keyboard bell rings and selection continues.
- The CANCEL key moves the selector to the default option if defined, otherwise to the rest position.
- The DO key displays the frame associated with the selected option if option selection was successful. Otherwise, it rings the keyboard bell.

DESIGNING A HELP STRUCTURE

- The HELP key rings the keyboard bell. In order to continue through the help structure, the user must select an option.
- The HOLD SCREEN key functions normally.
- The INTERRUPT key does not by itself return control to your application. If followed by the DO key, the system aborts your application and displays the P/OS Main Menu. Otherwise, option selection continues as usual.
- The PREV SCREEN key displays the previous help frame, if defined. Otherwise it rings the keyboard bell.
- The PRINT SCREEN key functions normally.
- The RESUME key returns control to the menu from which the user entered the help structure or, if the help menu was activated directly by your task, returns control to it.
- Invalid keys ring the keyboard bell but do not return control to your application.

8.4.3 Help Text Frames

Help text frames can be a full frame (16 lines of text), the top half of the screen (eight lines), or the bottom half of the screen (eight lines). Figure 8-6 shows a help text frame on the bottom half of the screen. When the user pressed the HELP key, the selector was positioned on the option "Reading the Report Card" in Figure 8-5.

When displayed implicitly by POSRES, half-screen frames do not alter the remainder of the screen. For example, in 8-5, part of the menu remains visible. If, however, your task calls HELP directly to display a half-screen frame, POSRES clears the entire screen.

It is recommended that you use half-screen frames for menu frame pointers when the size of the help text permits. Choose top or bottom in order to allow relevant areas of the previous frame to remain visible. You can design the help structure so that top and bottom frames alternate, allowing the user to see two help frames at a time.

8.4.4 Key Processing in Help Text Frames

While a help text frame is active, only the following keys are processed:

DESIGNING A HELP STRUCTURE

- The HELP key and the NEXT SCREEN key display the next help frame, if one was defined when the frame was created. Otherwise, the keyboard bell rings.

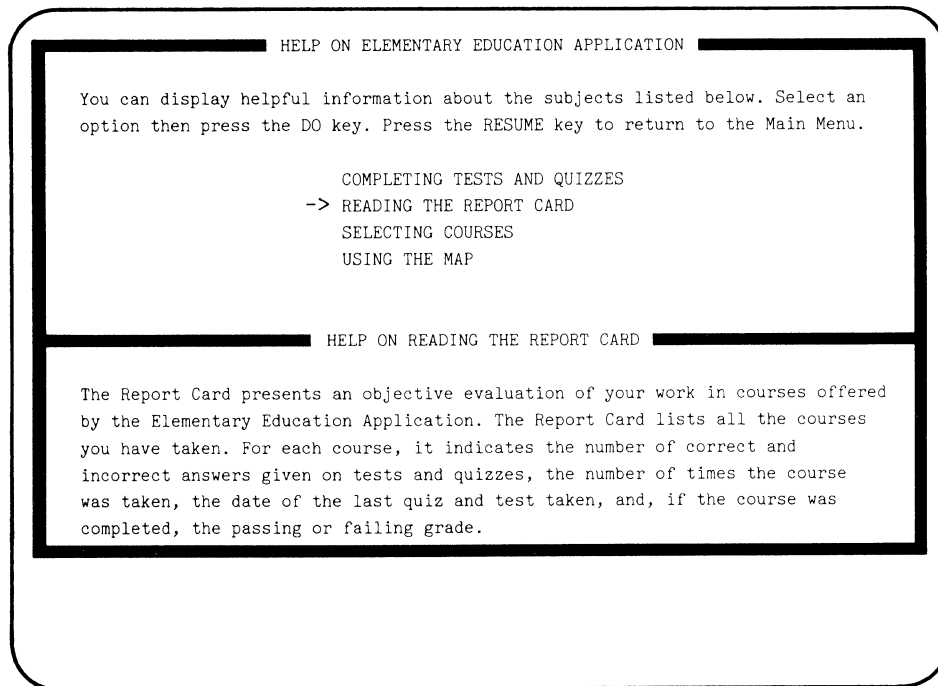


Figure 8-6: Help Text Frame

- The PREV SCREEN key returns to the previous help frame, if defined. Otherwise, the keyboard bell rings.
- The RESUME key returns control to the menu from which the user entered the help structure or, if the help menu was activated directly by your task, returns control to it.

8.4.5 A Sample Help Structure

Figure 8-7 shows part of the P/OS V1.7 Main Menu help structure in diagram form. In mathematical terms, the structure is a directed graph with nodes representing frames and edges representing single keystrokes and groups of keystrokes that select options.

The node labeled "HELP MENU" actually has several other options associated with it. Only the part of the graph resulting from the "Function Keys" option is shown.

DESIGNING A HELP STRUCTURE

This help structure could be changed significantly by altering some frame descriptions. For example, only "Frame 1" has a return path to the "Help for Function Keys" menu. On the other "Frame n" nodes, the previous screen is defined to be the next lower "Frame n" node.

You could make all of the "Frame n" nodes return to the "Help for Function Keys" menu. You also could modify "Frame 5" so that the NEXT SCREEN key goes to "Frame 1" forming a ring. That way, a user could cycle forward through the frames and still be able to back out at any frame, without returning to the Main Menu.

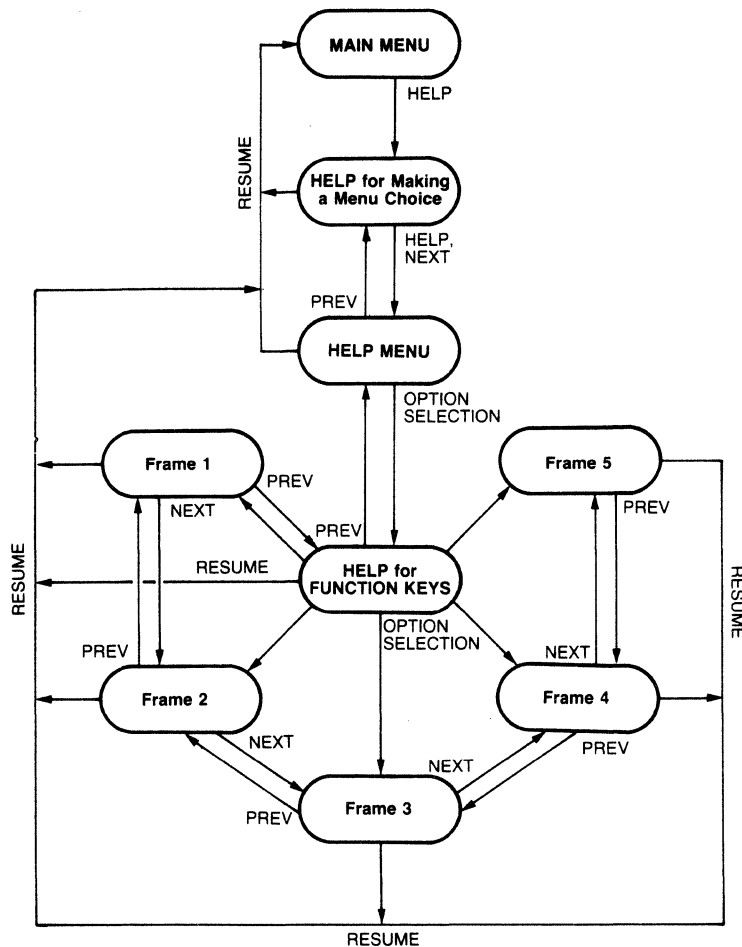


Figure 8-7: The P/OS Main Menu Help Structure (partial)

IMPLEMENTING A HELP STRUCTURE

8.5 IMPLEMENTING A HELP STRUCTURE

Help structures are made up of frames and frame pointers. A frame pointer is data that specifies a help menu or help frame by name. For static menus, you specify the frame pointers on the FDT Profile and Action forms. For dynamic menus, you specify frame pointers in the parameters passed to the POSRES routines that create the menu.

All menus contain a global frame pointer. When the user presses the HELP key with the pointer in the rest position, POSRES activates the help structure and displays the frame specified by the global frame pointer.

Single-choice menus (static or dynamic) contain a frame pointer for each option on the menu. When the user presses the HELP key with the pointer on an option, POSRES activates the help structure and displays the frame specified by the option frame pointer.

POSRES also maintains a default help frame pointer for use when no menu is active or when the current menu does not define a help frame. Your application can explicitly activate a help structure and/or specify a new default help frame pointer.

8.5.1 Opening Help Files

There are two ways to open a help definition file:

- The ASSIGN HELP command in your installation command file (see the Tool Kit Reference Manual) opens a help file at run-time. If your application uses only one help file, that's all you have to do.
- The Open Help File (HFILE) routine explicitly opens a help file. To use it, however, you must specify the name of the directory that contains the menu file. For a P/OS Hard Disk application, that requires translating the logical name APPL\$DIR (see Section 6.3).

POSRES assumes that all frames used by the current help structure are in the most recently opened help file.

8.5.2 Setting the Default Help Frame

There are several ways to set the default help frame:

IMPLEMENTING A HELP STRUCTURE

- The ASSIGN HELP command in your installation command file (see the Tool Kit Reference Manual) specifies the default help frame.
- The Open Help File (HFILE) routine specifies a default help frame. This takes precedence over a default help frame specified in your installation command file.
- The Specify Help Frame (HFRAME) routine specifies the default help frame. This takes precedence over a default help frame specified in your installation command file.
- Each time the user presses the DO key to select a menu option, that option's frame pointer, if defined, becomes the default. This automatically maintains the help context.

8.5.3 Activating the Help Structure

While a menu is active, POSRES automatically activates the help structure using the current menu's frame pointers whenever the user presses the HELP key. This help processing is entirely transparent to your application. POSRES uses the current menu's frame pointers in the following order of precedence:

- The option help frame (single-choice menus only).
- The global help frame.
- The default help frame.
- If there is no default help frame, POSRES simply refreshes the current frame and continues.

While no menu is active, you can activate the help structure by calling the Display Help Frame (HELP) routine. POSRES uses frame pointers in the following order of precedence:

- The help frame specified in the HELP call.
- The default help frame.
- If there is no default help frame (it is invalid) an error occurs.

You can also use the HELP routine while a menu is active. For example, if your application detects an inordinate number of user errors, it can suspend menu operations and activate the help structure.

MESSAGE FILES AND SERVICES

8.6 MESSAGE FILES AND SERVICES

Like frame definition files, message definition files are created with FDT. Message frames can contain up to 21 lines of text. By removing the string constants from your task and placing them into a message definition file, you can reclaim a significant amount of virtual memory.

Unlike frame definition files, POSRES provides no routines to explicitly display message frames. It does, however, provide a way to extract message frames as needed so that you can use them as parameters to other routines, such as the optional text lines on menus and those routines listed below.

For example, the Installation Verification Program used by the Tool Kit languages uses that technique for its Geography test answers. Figure 8-8 shows the same multiple-choice menu in Figure 8-3 with a message informing the user of an incorrect selection. This was accomplished by extracting message frames and using them to specify the optional text lines in the POSRES calls that created the menu.

COURSE: GEOGRAPHY OF NORTH AMERICA

This course covers rivers, lakes, and mountains. The first topic is mountains. The mountains listed below are located in different parts of the world. Select all the mountains located in North America:

- > MT. RUSHMORE
- > MT. WASHINGTON
- > MT. HOOD
- > GRAND TETON
- > MT. RAINIER
- MT. FUJI
- > KILIMANJARO
- > MT. ST. HELENS
- >-> MT. MCKINLEY

Choose one or more options with the SELECT key and press the DO key:

Kilimanjaro is not in North America. It is in Tanzania, Africa.
At 19,340 feet, it is the highest point in Africa.

Figure 8-8: Message Frame

POSRES includes the following message service routines:

MESSAGE FILES AND SERVICES

- **Fatal Error (FATLER)**

This routine provides a consistent way to inform the user of a fatal error condition. It blanks line 22, displays the message "Application error. Press RESUME to return to Main Menu." on line 23, and displays user-supplied text on line 24. That text can be a message that tells the user why the application failed and where to look for recovery information.

- **Read Message (RDMSG)**

This routine reads a message from a specified message file into a buffer. You do not have to explicitly open the message file; POSRES opens it for you each time you call RDMSG. If either the file or the frame identifier specified in the RDMSG call cannot be opened or located, POSRES fills the buffer parameter with the message "Can't access filename or can't find frame frameid."

- **Send Message to Message/Status Display (MSGBRD)**

This routine sends a message to the P/OS Message/Status Display, which can be viewed by selecting the "View Message/Status" option on the P/OS Main Menu. The Main Menu shows how many unread messages have been queued. When the user selects "View Message/Status", P/OS displays the messages in the order in which they arrived.

NOTE

The MSGBRD routine is not actually in POSRES; it is in the system library (SYSLIB). If you use the MSGBRD routine, you must also edit your Application Builder command file as described in Section 8.8.

8.7 FUNCTION KEYS

The Professional keyboard contains three types of function key: reserved, prelabeled, and generic. The reserved function keys, F1 and F2 (also known as HOLD SCREEN and PRINT SCREEN), are accessible only to the terminal subsystem. The other function keys are accessible to applications. All of the function key codes and labels are shown in Appendix D.

FUNCTION KEYS

8.7.1 Using Function Keys

The basic Professional keyboard comes with 12 prelabeled function keys and 16 generic function keys (F3 to F14, and F17 to F20). The prelabeled keys are:

- arrows (4)
- Do
- Find
- Help
- Insert Here
- Next Screen
- Prev Screen
- Remove
- Select

P/OS provides a keyboard label strip that assigns semantic meanings to 11 of the generic keys:

- Addtnl Options
- Break
- BS
- Cancel
- ESC
- Exit
- Interrupt
- LF
- Main Screen
- Resume
- Set-Up

If your application provides its own keyboard label strip, you can assign your own semantic meanings to the keys. Otherwise, use the meanings defined by the P/OS label strip, as described below.

- Use ADDTNL OPTIONS to accept auxiliary commands, and to offer services beyond those offered in the current context.
- Use the arrow keys to move the cursor (or pointer) around the screen in order to select or place objects. For example, a spreadsheet could use the arrow keys to move to an adjacent cell. In a menu tree, the up and down arrow keys are used in option selection.
- Use BREAK only in communications applications to transmit a break character.
- Use DO and CANCEL to confirm and reject input, respectively. Use DO to indicate that a choice has been made or a value entered and that the user is ready to proceed. Use CANCEL to indicate that the choice or value is incorrect and must be reentered. In a menu

FUNCTION KEYS

tree, these keys are used in option selection.

- Use EXIT to return to the point from which the current activity was invoked. In a menu tree, use it back up one menu. If at the outermost level, use EXIT to terminate the application.
- Use the INTERRUPT/DO sequence to request "emergency" termination of your application. You can process INTERRUPT/DO yourself or allow P/OS to process it for you.
- Use MAIN SCREEN to return to internal command level. In a menu tree, use it to jump to the "root node". If already at this level, use MAIN SCREEN to terminate the application.
- Use NEXT SCREEN and PREV SCREEN to move through a series of displays. For example, PROSE uses them to move through a document. In a help structure, they activate frame pointers.
- Use RESUME to indicate that the user is ready to continue an interrupted activity. In a help structure, RESUME returns control to the menu or task that activated the structure.
- Use SELECT to record the current position of the cursor (or pointer) for some subsequent action. In a multiple-choice menu, SELECT is used in option selection.

8.7.2 Programming Function Keys

When a function key (other than DO) terminates a menu, control returns to your application with the following values in the status block:

first word	second word
-14	key code

POSRES also provides several routines to help you process function keys independently from the menu interface. They are:

- **Get Keystroke (GETKEY)**

This routine inputs a single keystroke from the terminal without echo. The first word of the status block contains one of the following values:

- +1 indicates that the user pressed a main keyboard key. The second word contains the DEC Multinational decimal code of the key.

FUNCTION KEYS

+2 indicates that the user pressed a function key. The second word contains one of the codes listed in Appendix D.

n indicates that an error has occurred. The second word contains one of the error codes listed in Appendix C.

- **Parse String (PRSCSI)**

This routine parses a string for a control sequence (CSI). It scans the string from the left for a CSI character, returns its position in the string, and translates the subsequent characters into a code representing one of the function keys shown in Appendix D.

- **Wait for Resume Key (WTRES)**

This routine echoes all keystrokes except the RESUME key by ringing the keyboard bell. When the user presses the RESUME key, control returns to your application. You can use this routine to allow the user to read something on the screen or change a diskette, for example, before proceeding. Before calling WTRES, display a message such as "Press RESUME to continue." on the screen.

8.8 POSRES TASK IMAGE REQUIREMENTS

POSRES requires that you make some edits to your PAB command file. To determine which edits are required, make a list of the POSRES routines used by your task. Subsequent sections will describe how to edit the .CMD file.

NOTE

Be sure to include any POSRES routines that are used by your high-level language run-time support library (see your language documentation and Chapter 7).

Figure 8-9 shows a sample .CMD file, written for a PASCAL program that uses all of the POSRES routines. The contents of this file are explained in detail in the following sections. It is assumed that you are familiar with the general contents and purpose of a .CMD file. If not, please turn to Chapter 7.

```
PASDEM/CP/FP,PASDEM/MA/-SP=PASDEM/MP
CLSTR=PASRES,POSRES,RMSRES:RO
STACK  = 30                ; Startup stack size
UNITS   = 46                ; Number of units available
GBLDEF  = TT$EFN:7          ; Terminal I/O event flag number
ASG     = LB:33:34:35:36    ; System device
```

POSRES TASK IMAGE REQUIREMENTS

```
GBLDEF = MS$LUN:41      ; Message frame file
GBLDEF = MN$LUN:42      ; Menu Frame file
GBLDEF = MB$LUN:43      ; Message/Status display
GBLDEF = HL$LUN:44      ; Help frame file
ASG     = SY:37         ; P/OS current device
GBLDEF = WC$LUN:45      ; OLDFIL/NEWFIL device
ASG     = TI:38         ; User terminal
GBLDEF = TT$LUN:46      ; Terminal I/O
EXTSCT  = MN$BUF:4540   ; Static single-choice buffer
EXTSCT  = DM$BUF:4540   ; Dynamic single-choice buffer
EXTSCT  = MM$BUF:1000   ; Multiple-choice buffer
EXTSCT  = HL$BUF:3410   ; Help frame buffer
EXTSCT  = FL$BUF:4310   ; OLDFIL/NEWFIL buffer
//
```

Figure 8-9: PAB Command File with POSRES Options

8.8.1 The UNITS Option

The UNITS option specifies (in decimal) how many logical units (LUNs) your application requires. LUNs identify simultaneously open files or devices.

8.8.2 The GBLDEF Option

The global symbol definition (GBLDEF) option equates a symbolic name to an octal number. POSRES requires a GBLDEF option for each symbol shown in Table 8-1. Do not omit any symbol.

Compare the list of POSRES routines used by your task to Table 8-1. Some POSRES routines do not require symbols and are not shown in the table. Equate a logical unit number (as described in Section 8.8.3) or an event flag number (described below) to each symbol associated with each POSRES routine on your list. For the remainder of the symbols, specify LUN zero. For example, if your task does not call MSGBRD, you can specify:

```
GBLDEF = MB$LUN:0
```

Most POSRES routines require an event flag (EFN) to perform terminal I/O. Select an event flag number in the range 1 to 24 decimal, convert it to octal, and use the GBLDEF option to equate the symbol TT\$EFN to the octal number as shown in Figure 8-9. This assignment must not conflict with any other event flag assignments (language run-time systems use DIGITAL-reserved event flags 25-32).

POSRES TASK IMAGE REQUIREMENTS

Table 8-1: POSRES Global Symbols

Routine	HL\$LUN	MB\$LUN	MN\$LUN	MS\$LUN	TT\$EFN	TT\$LUN	WC\$LUN
DMENU	X	-	-	-	X	X	-
FATLER	-	-	-	-	X	X	-
GETKEY	-	-	-	-	X	X	-
HCLOSE	X	-	-	-	-	-	-
HELP	X	-	-	-	X	X	-
HFILE	X	-	-	-	-	-	-
HFRAME	X	-	-	-	-	-	-
MCLOSE	-	-	X	-	-	-	-
MENU	X	-	X	-	X	X	-
MFILE	-	-	X	-	-	-	-
MFRAME	-	-	X	-	-	-	-
MMENU	X	-	-	-	X	X	-
MSGBRD	-	X	-	-	-	-	-
NEWFIL	-	-	X	-	X	X	X
OLDFIL	-	-	X	-	X	X	X
PRSCSI	-	-	-	-	-	-	-
RDMSG	-	-	-	X	-	-	-
WTRES	-	-	-	-	X	X	-

X = symbol used
- = symbol not used

8.8.3 The ASG Option

The ASG option associates a physical device with one or more logical unit numbers (LUNs). These assignments tell POSRES which devices to use. For example, your application's frame definition files reside on device LB:, the P/OS system device. Without the assignments, POSRES would look for these files on SY:, the P/OS current device. On a P/OS Hard Disk system, that is usually, but not necessarily, identical to the hard disk, LB:. On a P/OS Diskette system, however, SY: is the drive with the data diskette, and LB: the drive with the application diskette.

The ASG option accepts decimal numbers but the GBLDEF option accepts octal numbers. To avoid confusion, use this procedure to assign LUNs:

1. Use the ASG option to associate a device with a decimal LUN.
2. Convert the LUN from decimal to octal.

POSRES TASK IMAGE REQUIREMENTS

3. Use the GBLDEF option to equate a symbol to the octal LUN.

Using available LUNs in the range 1 to 128, make the following assignments.

NOTE

Your assignments must not conflict with any other ASG assignments in your .CMD file.

- Assign four LUNs to LB:, the device that contains the application directory, and equate them to MS\$LUN, MN\$LUN, MB\$LUN, and HL\$LUN. For example, if you want to use LUNs 33 through 36 and they are not used elsewhere, insert:

```
ASG      = LB:33:34:35:36
GBLDEF   = MS$LUN:41
GBLDEF   = MN$LUN:42
GBLDEF   = MB$LUN:43
GBLDEF   = HL$LUN:44
```

- Assign a LUN to SY:, the P/OS current device, and equate it to WC\$LUN. For example, if you want to use LUN 37 and it is not used elsewhere, insert:

```
ASG      = SY:37
GBLDEF   = WC$LUN:45
```

- Assign a LUN to TI:, the Professional keyboard/video monitor, and equate it to TT\$LUN. For example, if you want to use LUN 38 and it not used elsewhere, insert:

```
ASG      = TI:38
GBLDEF   = TT$LUN:46
```

8.8.4 The EXTSTCT Option

POSRES uses the program section names shown in Table 8-2 as buffers to store menus, help frames, and so forth. Insert an EXTSTCT option for each buffer shown to extend its program section by a specified (octal) number of bytes.

Compare your list of used POSRES routines to Table 8-2 and assign each accessed buffer sufficient size, as described below. If a routine does not appear in the table, it does not access any buffer. If your task does not use any of the routines that access a particular buffer, assign that buffer a size of zero. Do not omit any buffer names.

POSRES TASK IMAGE REQUIREMENTS

Table 8-2: Buffers Accessed by POSRES Routines

Routine	DM\$BUF	FL\$BUF	HL\$BUF	MM\$BUF	MN\$BUF
DMENU	X	-	X	-	-
DPACK	X	-	-	-	-
HCLOSE	-	-	X	-	-
HELP	-	-	X	-	-
HFILE	-	-	X	-	-
HFRAME	-	-	X	-	-
MCLOSE	-	-	-	-	X
MENU	-	-	X	-	X
MFILE	-	-	-	-	X
MFRAME	-	-	-	-	X
MMENU	-	-	X	X	-
MPACK	-	-	-	X	-
MUNPK	-	-	-	-	X
NEWFIL	-	-	X	-	X
OLDFIL	-	X	X	X	X

X = buffer accessed
 - = buffer not accessed

Figure 8-10 shows an example of the EXTST options with values calculated for the largest possible frame of each type. In other words, if every field on a menu, help, and message frame were filled in with the suggested maximum amount of data, the respective buffers would have to be allocated the sizes shown.

```

EXTST = DM$BUF:4540    ; Dynamic single-choice buffer
EXTST = FL$BUF:4310    ; OLDFIL/NEWFIL buffer
EXTST = HL$BUF:3410    ; Help frame buffer
EXTST = MM$BUF:1000    ; Multiple-choice buffer
EXTST = MN$BUF:4540    ; Static single-choice buffer
  
```

Figure 8-10: Suggested Maximum POSRES Buffer Sizes

NOTE

There is no maximum size for FL\$BUF or MM\$BUF. If you use extraordinarily large menus, you might have to expand these buffers.

It is recommended that you begin with the maximum size for each accessed buffer. If your task exceeds the bounds of virtual memory, you can reduce the size of one or more buffers as described below.

POSRES TASK IMAGE REQUIREMENTS

Some care should be exercised when shrinking POSRES buffers. If you change a frame definition file, you might find that a frame has become too large for its buffer, requiring you to enlarge the buffer and task build again. For example, translating frames into another language can cause them to expand. Therefore, allocate some extra space to any buffer for which the maximum frame size is not completely stable.

To compute the minimum buffer for a static display (a static single-choice menu, help menu, and so forth):

1. Take the size of the largest frame (provided by FDT in decimal when you use the CONVERT command). If the source data file is unavailable, you can use a system utility to analyze the frame definition file. On a VAX/VMS system, type:

```
$ ANALYZE/RMS file.typ
```

The longest record in the file will appear under the "RMS FILE ATTRIBUTES" section of the output. On an RSX-11M/M-PLUS system or on the PRO/Tool Kit, type:

```
$ RUN $DMP
DMP> TT:=filename.typ/HD/BL:0
```

The length of the longest record will appear in the F.RSIZ field in the "HEADER AREA" of the output.

2. Add 200 bytes to this number.
3. Convert the result to octal.
4. Use this number with the EXTSTCT option.

You can approximate the minimum buffer size for a dynamic single-choice menu by using FDT to build a similar static single-choice menu and using its converted size. Choose a frame that represents your worst (largest) case situation and allow a large margin for error (approximately 200 bytes). Otherwise, use this procedure:

1. Total the sizes of all the fields in the largest frame.
2. Add an overhead of eight bytes per field.
3. Add 200 bytes to the total.
4. Convert the result to octal.

POSRES TASK IMAGE REQUIREMENTS

5. Use this number with the EXTSTCT option.

The minimum buffer size for a multi-choice menu is difficult to determine because it depends on factors present at run-time. If you find it necessary to reduce this buffer, the recommended procedure is: change the buffer size by the desired amount task build, and test your application thoroughly, using the largest possible multiple-choice menu.

NOTE

The following section assumes that you are familiar with Overlay Descriptor Language. If not, it is recommended that you refer to the RSX-11M/M-PLUS Task Builder Manual before continuing.

8.8.5 Placing Buffers in Overlay Branches

Another way to reclaim virtual memory is to place all code that references POSRES routines and buffers into overlay segments. Unless you specify otherwise, the PAB allocates POSRES buffers in the root.

If you put a POSRES buffer in an overlay segment:

- The buffer's contents will be reinitialized whenever that segment is loaded into memory. Thus, you are responsible for replacing its contents.
- You cannot call the POSRES routine that uses that buffer from another overlay segment (except from another co-tree). Thus, you must subdivide your task into segments that use POSRES and those that do not.

Figure 8-11 shows an example of how to place the buffers into an overlay branch.

POSRES TASK IMAGE REQUIREMENTS

```

        .ROOT      RCODE-RMSROT-( *MENU,*CODE)
MENU:    .FCTR      MCODE-BUFFA-BUFFB
CODE:    .FCTR      ARTN-BRTN-CRTN
@LB:[1,5]RMSRLX

        .PSECT     DM$BUF,RW,D,GBL,REL,CON
        .PSECT     DM$BUG,RW,D,GBL,REL,CON
        .PSECT     FL$BUF,RW,D,GBL,REL,CON
        .PSECT     FL$BUG,RW,D,GBL,REL,CON
        .PSECT     FL$FAB,RW,D,GBL,REL,CON
        .PSECT     HL$BUF,RW,D,GBL,REL,CON
        .PSECT     HL$BUG,RW,D,GBL,REL,CON
        .PSECT     MM$BUF,RW,D,GBL,REL,CON
        .PSECT     MM$BUG,RW,D,GBL,REL,CON
        .PSECT     MN$BUF,RW,D,GBL,REL,CON
        .PSECT     MN$BUG,RW,D,GBL,REL,CON

BUFFA:   .FCTR      DM$BUF-DM$BUG-FL$BUF-FL$BUG-FL$FAB-HL$BUF
BUFFB:   .FCTR      HL$BUG-BM$BUF-MM$BUG-MN$BUF-MN$BUG
        .END

```

Figure 8-11: Sample .ODL File Showing Overlaid Buffers

In Figure 8-11, the modules RCODE, MCODE, ARTN, BRTN and CRTN all refer to user-supplied portions of the task. Use the .PSECT directives exactly as shown (if they do not match the actual attributes of the PSECTs that they reference, you will get errors when task building).

The POSRES buffers that you can overlay are:

<u>Buffer</u>	<u>Use</u>
DM\$BUF, DM\$BUG	Dynamic menu
FL\$BUF, FL\$BUG	OLDFIL, NEWFIL
FL\$FAB	OLDFIL, NEWFIL
HL\$BUF, HL\$BUG	Help frames
MM\$BUF, MM\$BUG	Multi-choice menu
MN\$BUF, MN\$BUG	Static, single choice menu

The xx\$BUF/xx\$BUG pairs must be in the same overlay. These pairs of PSECTS are used by POSRES to determine buffer sizes. If they are not in the same overlay, unpredictable behavior will result.

CHAPTER 9

APPLICATION TUNING

Application tuning consists of the reduction of unnecessary operations and the use of system specific knowledge allowing the more efficient use of features. An application for P/OS should make as much use as possible of the facilities provided by the Professional 300, both hardware and software. While there are no requirements that an application use these facilities they were designed with the idea of a user-friendly, integrated atmosphere with consideration towards easing the job of writing an application.

9.1 SYSTEM DIRECTIVES

It is in the best interests of applications to use the least expensive directive for a given purpose. From the larger view of the application, a system directive is relatively inexpensive; but from a smaller view at the instruction level a system directive takes many instructions to process. This adds to the overhead of the task and contributes to any performance problems that might be observed.

9.1.1 Cost of Directives

When comparing directives it will be sometimes be found that there are several directives, or combinations of directives, that will accomplish a given function. Of these directives, one is probably the least expensive to use in terms of system overhead. Which is the least expensive is not always easy to ascertain, but the determination is usually worthwhile. Therefore, the goal is to make use of the directive or directives that incur the least amount of system overhead.

As an example, the PLAS (memory management) directives vary greatly in execution time and complexity. For instance, the EXTK\$ directive will be very expensive when it is necessary to checkpoint the task region to disk and reallocate memory in the system controlled "GEN"

SYSTEM DIRECTIVES

partition. The MAP\$ directive is cheaper than UMAP\$/MAP\$ pairs, which are cheaper than the ELAW\$/CRAW\$ pairs. Other directives have ways in which they can be made less expensive.

9.1.2 Directives .vs. Servers

There is a small area of overlap between callable system routines and several system directives. The logical name support directives and the default directory directives can both be manipulated by a callable system routine or directly by system directives. If you are programming in a language that allows the issuing of system directives then you may want to consider issuing the directives rather than calling the system routine.

Let's examine the specific cases. For the logical name directives CLOG\$, DLOG\$, and TLOG\$ the system routine PROLOG translates the strings to upper case and issues the system directive to perform the function. The task could simply issue the directive itself and save a large amount of time. In this instance, the task must specify LT.USR and a modifier of 0 and ensure that the logical is uppercased if used by RMS-11 or another task or system service. Logical strings are binary and hence case sensitive. This is why the server always translates to upper case first.

Otherwise, if the callable routine is used, the task that services the request must be loaded into memory and started, it must receive and process the request via a system directive and then return the results to the calling task. For logical name support, the additional services provided by the callable routine are to ensure that the string is in upper case and specify LT.USR and modifier 0.

Default directory support on the other hand should be done using the server. The server actually performs RMS parsing operations on the input string to determine that it contains a valid device and directory specification. If the task chooses to issue the directives itself, there is a possibility that it could specify an invalid directory string. The PROLOG call uses the RMS parser to verify that the device syntax is legitimate. The directive, on the other hand, accepts any string as the default directory string. This could cause problems when attempting to find or create files. So, in this case it is best if the application task uses the POSSUM PROLOG routine to process the request despite the extra overhead.

9.2 FILE HANDLING

When dealing with files, there are several methods that may be used to increase application performance. Almost all of these methods require

FILE HANDLING

some trade-offs, either in terms of some additional logic in the task or in the use of disk space.

9.2.1 When to Open Files

One of the simplest techniques to use when handling files is to defer the opening of any file until it is necessary. It may at first seem better to open all files at application startup time but the primary tuning goal at application startup is usually to start interacting with the user as soon as possible. Opening files takes time and if at all possible should be put off until data is actually needed from the file. This should not pose too many difficulties for the task.

9.2.2 Use of File ID's

The use of file ID's for operations on files reduces overhead. If possible, preserve the file ID of a file that is to be opened several times. This incurs less overhead than finding the file by name each time. The cost to your application is merely to remember the file ID. It would be even more efficient to simply keep the file open.

9.2.3 File Pre-Allocation

Preallocating the space necessary for a file will also save time and cut overhead. If a file does not contain sufficient room to append records, then the necessary disk space must be allocated to the file. By preallocating the amount of space that the file will require at the time the file is created your task can avoid the overhead of later extending the file.

The technique of preallocating sufficient space will only work in instances where the size of the file to be written is known in advance. This is not always the case. However, if it is known that a file will require even a minimum amount of space, then preallocating that minimum amount will save time later.

9.2.4 Pre-extending

An additional method by which to improve file performance is to specify a reasonable default extend quantity to RMS. This means that whenever RMS needs to extend your file to add an additional record it will extend the file by the number of blocks specified in the default extend quantity. If this number is reasonably large, this will result

FILE HANDLING

in fewer extensions to your file which will translate into improved performance for your application.

Prior to version 1.7 of P/OS, if you did not specify a default extend quantity for sequential files, you received only enough extension to satisfy the I/O request. With version 1.7, if you do not specify a default extend then the file is extended by at least 10 blocks whenever an extension is necessary. This improves performance by a significant amount. This increase in performance may be improved by experimenting with the default extend quantity such that your application works at its best. Note that RMS will truncate the file to its logical EOF if the last extend was an implicit extend. That is, if no extensions occurred or the last extend was caused by the task asking RMS to explicitly extend it, then the file will not automatically be truncated on close.

9.2.5 Multiblock I/O

As with most I/O operations, the more data transferred per operation the more efficient the operation becomes. This applies to disk I/O whether it be reads and writes directly to the disk or through an intermediate record processor such as RMS. Increasing the size of the reads and writes will reduce the time per block spent reading or writing buffers to the disk as well as your virtual address space.

9.3 VIDEO PERFORMANCE

Video performance is perhaps one of the most sensitive areas in terms of performance and tuning. No matter how fast an application may be, if the video throughput is perceived as slow then that application will be thought of as slow. For this reason it is worth spending some time on an application to insure that the video performance is as high as possible. There are a number of simple rules and techniques that can be applied to an application to help tune for the best possible throughput.

9.3.1 Size of Buffer

The size of the output buffers is one of the most important factors governing throughput rate. Very simply, the larger the buffers the less time spent in overhead and the greater the throughput rate. Above 64 characters per buffer the output rate becomes asymptotic to the current maximum throughput rate of the video.

VIDEO PERFORMANCE

9.3.2 Buffering

One of the largest possible improvements in terminal output can be obtained from the use of large buffers. One method that may be used to increase the size of the output buffers for the terminal is the use of an intermediate buffering scheme. This buffering scheme employs a routine that accepts text to be output to the screen. This text is then placed into an intermediate buffer. When this intermediate buffer is full it is output to the terminal. A secondary routine to force output to the terminal is also provided to allow the forced flushing of the intermediate buffer.

Let's take a look at a specific example of this. Assume a task generates output of varying lengths and at varying times. Not all output strings generated by this task are terminated by a carriage-return, line-feed pair (a prompting string, for example). Following is a listing of the routines necessary to handle the intermediate buffering:

```
; OUTCHR -
;   This routine is used to output a character to the terminal.
; All output is temporarily stored in an intermediate output buffer
; before queuing to the terminal. Note that this routine assumes
; explicit carriage control in the text strings.
;
; Input:
;           R0 - Character

Outchr: Dec     Ttyctr           ; Decrement the count
        Blt     Outch0          ; No room left, make some
        Movb    R0,@Ttyptr      ; Store the byte
        Inc     Ttyptr          ; Increment pointer
        Return                    ; Then return to caller

Outch0: QioW$C   Io.Wvb,Tt$Lun,Tt$Efn,,,,<Ttybuf,.Lntty,0>
        Mov     #.Lntty,Ttyctr  ; Reset the counter
        Mov     #Ttybuf,Ttyptr  ; and the pointer
        Br      Outchr          ; Then try again to output the char.

; STRING -
;   This routine is used to print an ASCIIZ string on the terminal.
;
; Input:
;           R0 - Pointer to string

String: Call     $Saval          ; Save registers
        Mov     R0,R5           ; Save up pointer
Strin0: Movb     (R5)+,R0        ; Get next byte
        Beq     Strin1          ; Done
        Call    Outchr          ; Output the character
        Br      Strin0          ; Loop
```


VIDEO PERFORMANCE

```
Strin1: Return                ; Return to caller

; FRCOUT -
; This routine is used to force out the TTY buffer

Frcout: Mov    #.Lntty,R0      ; Get buffer length
        Sub    Ttyctr,R0      ; Compute amount of buffer filled
        Beq    Frcou0         ; None, just exit
        QioW$S #Io.Wvb,#Tt$Lun,#Tt$Efn,,,,<#Ttybuf,R0,#0>
        Mov    #Ttybuf,Ttyptr ; Reset pointer
        Mov    #.Lntty,Ttyctr ; and counter
Frcou0: Return                ; Return to caller

; Terminal buffer

        .Even
Ttyctr: .Blkw  1                ; Buffer byte count
Ttyptr: .Blkw  1                ; Byte pointer
Ttybuf: .Blkb  80.             ; TTY buffer
        .Lntty = .-Ttybuf      ; Length of buffer
```

The code shown above will buffer up all text until the buffer is full or the routine to force output is called. Circumstances under which the force output routine might be called are:

- At the end of any escape sequence to prevent it from being broken into two separate QIO's.
- In the OUTCHR routine whenever a carriage-return or line-feed character is detected. This may seem to violate the rule of buffering as much as possible, but forcing out buffers on line terminator characters may make the output look less clumpy to the user.
- In an AST routine specified in a MRKT\$ directive which is called a few times a second to dump the buffer. If text data is showing up at random intervals and without line terminators to rely on, this may be the only way to output a sequence that does not end with a line terminator, such as a prompting sequence. The rate at which the AST routine would be called is dependent on the rate at which data shows up and with consideration given to the amount of overhead caused by the AST routine.

9.3.3 Eight Bit Escape Sequence Characters

The use of the eight bit characters in escape sequences will decrease the number of characters per sequence by one. Replacing the escape-[sequence with the CSI character is one example. This may not seem like much but it reduces the amount of work that must be done by

VIDEO PERFORMANCE

several different components. This, in turn, translates into more CPU time for your application task as well as slightly greater video throughput. Offsetting this advantage is the fact that VT100 support is lost.

9.3.4 QIO\$.vs. QIOW\$

All text output to the video on the Professional is handled by a task that actually draws each character onto the bitmapped screen. For a variety of reasons an asynchronous QIO will not be returned to the task until very near the completion of the actual I/O. Even if it were, the terminal task runs at a very high priority and would lock out the application task. Therefore, there is no great advantage to using asynchronous QIO's over a synchronous QIOW\$ on the Professional. So, if you are writing a new application that uses QIO's to the video, then you might want to use the QIOW\$. As noted, there is not a large advantage to using asynchronous QIO's to the video, and additionally the task will be simpler overall with synchronous QIO's. The preceding discussion does not apply to other devices such as the printer port (TT2:).

9.3.5 Turning the Cursor Off

Another way to save system overhead and increase the amount of CPU time available to your task is to simply turn the cursor off via the appropriate escape sequence. The terminal task will no longer have to blink the cursor and this will save time for your application. Note that this technique should not be used in every possible instance. Consideration must be given to how the screen will appear without a cursor. For some applications, such as a spreadsheet, the lack of a blinking cursor may not be a problem. For other applications turning off the cursor may startle the user and possibly even make it appear that the machine has crashed.

Now consider this from a wider perspective. The cursor, as far as the terminal task is concerned, is just a blinking field. The amount of time and overhead that is required of the system and the terminal task is large for the first blinking field. If there is more than one blinking field, the incremental overhead is not large. Therefore, if there are blinking fields on the screen turning the cursor off will not provide any substantial performance improvement. Before this technique is discarded because a pointer of some type is needed on the screen, consider using a non-blinking, reverse video field as a cursor substitute. Once again, this does not apply in every situation but may just do the trick for some applications.

VIDEO PERFORMANCE

9.3.6 Standard Video Tricks

There are several fairly standard "tricks" that can be used quite effectively when dealing with video output. The potential savings in performance and overhead can be quite large. Three of these techniques will be discussed and the cost of using them will be considered.

The first technique provides a very large potential savings in overhead. It is simply a matter of not writing more data than is necessary. For instance, your task manages the entire screen and provides updates only to certain portions of the screen. One method of updating the screen would be to rewrite the entire screen on every update. This would be rather slow as the amount of data could be quite large. A variation on this would be to only update any regions that are known to change dynamically. This is faster, but could still produce a large amount of output. What is needed are two copies of the dynamic sections of the screen image in your task. One copy reflects the current state of the screen, the other the desired state. Your task compares these memory-resident copies and determines which character positions must change and only updates those positions. This technique reduces the amount of I/O that must be performed to an absolute minimum, which in turn will increase your performance and terminal throughput.

The cost of this technique is in the virtual memory space required to contain two copies of any dynamic regions on the screen. Two copies of an 80 column, 24 line screen would require 3840 bytes. For some tasks, this may be too much. For others, the performance improvement would be well worth the added task size.

The second technique is used when the cursor position is changing rapidly and within a row or column. If this is the case, then the cursor motion escape sequences (such as UP, DOWN, LEFT, RIGHT) are less expensive than the use of the cursor positioning sequence. This method of moving the cursor works well in conjunction with the multiple screen technique. When combined they can result in tremendous savings in the amount of data that must be output and a corresponding increase in terminal throughput.

A corollary to this is to take advantage of the editing functions provided as part of the VT102 interface of the video. These include line insert, line delete, character insert, character delete.

The third technique also applies only to applications that manage the entire screen as well as producing large amounts of output. It is known that output which causes the screen to scroll will appear on the screen at a slower rate than output which does not cause the screen to scroll. This may be used to advantage when writing output to the screen. If the application produces logically distinct, single screens of output, then a performance improvement technique that can

VIDEO PERFORMANCE

be applied is to clear the screen and home the cursor before writing to the screen. This requires no penalty in the form of increased virtual address space, but consideration should be given to how the application will look and "feel" when screens of output are presented as a unit rather than as a single, scrolled amount of text.

9.4 KEYBOARD INPUT

Keyboard input is usually not considered as an area in which large improvements start. However, there are several methods by which keyboard input may be made more efficient as well as improving the "feel" of the application.

9.4.1 Function Keys

Perhaps the simplest method to improve the feel of the application is to make use of the function keys provided by the LK201 keyboard. These keys produce escape sequences by which the key may be identified and a specific action taken. Most applications should certainly make use of such standard keys as MAIN SCREEN, EXIT and HELP. Depending on the application, others may be translated into either functions such as NEXT SCREEN. Finally, for a custom touch the application may define and use unassigned function keys such as F17 through F20.

As mentioned in the preceeding paragraph, most applications should make use of the function keys. A suggested usage of the functions keys is provided in the appendix on P/OS User Interface Guidelines.

9.4.2 Eight Bit Characters

The use of eight-bit characters on input serves two purposes. The first use is to reduce the amount of overhead required to process any of the function, cursor or numeric keypad keys. The escape sequence generated by the keys will contain the eight bit equivalent character.

The second use for eight bit characters is that this mode will allow you access to the full multinational character set. More will be said about multinational characters later, but it is generally a good idea to keep your application free of any possible restrictions.

KEYBOARD INPUT

9.4.3 Input Buffering

As with terminal output buffering, input can also be made more efficient through the use of input buffering. Very simply, your task accepts input from the terminal and keeps it stored in an intermediate buffer until it is required by your task. This can be accomplished by polling the terminal at some interval or by attaching the terminal with AST notification of unsolicited input if single character input is needed by the application.

9.4.4 AST's With Notification

Consider the case of an application that requires keyboard input but not necessarily on a command line or even a line basis. One relatively efficient method of obtaining keyboard input in this case is by attaching the terminal device with AST notification of input. In this manner, whenever characters or function keys are typed on the keyboard they are delivered to your task en masse. That is, when you finally end up in your AST routine you may read in as many characters as occupy the typeahead buffer. This is more efficient than attaching with an AST per character. Also note that when a function key is typed the entire escape sequence generated by that function key is delivered to the terminal driver as a single sequence of characters. If this type of AST routine is used, the entire sequence may be read at one time from the typeahead buffer.

When in the AST routine be sure to issue the QIO\$ function SF.GMC, subfunction TC.TBF, to determine the amount of data that exists in the buffer. This will return the amount of data that should be read, which can then be input with a QIO\$ for the entire amount. The buffer may then be transferred to the actual input buffer (there may still be data in the task's internal input buffer, so this typeahead buffer that was just read in must be placed at the end of the data not yet processed).

9.5 PROGRAMMABLE LOGICAL ADDRESS SPACE (PLAS) CONSIDERATIONS

An application can benefit greatly from the use of the PLAS directives to modify its virtual address space or to create or map additional regions that contain data.

The most common traps that can develop are from misuse (or over-use) of the PLAS directives. Creating a single large data region could cause problems when attempting to bring it in and out of memory. Another example is in sending task regions by reference with the SREF\$ directive. The sender, receiver and all regions mapped by both must be in memory. This can cause severe memory contention and perhaps

PROGRAMMABLE LOGICAL ADDRESS SPACE (PLAS) CONSIDERATIONS

even a deadlock situation.

Other traps to watch out for are the use of some PLAS directives that can potentially waste system resources. Not using WS.NAT in the window block for a RREF\$ directive can waste system pool space and degrade performance. Other potential performance problems are not keeping a region mapped such that it is removed from memory and must be read in frequently; not using read-only or multi-user libraries where possible, to cut down on checkpointing time. There are certainly other problems that can arise, the watchword with PLAS directives is caution.

9.6 MULTI-TASK APPLICATIONS

Many applications seem to be multi-task applications. This may sometimes lead to special problems not encountered in a single task application. Most, if not all, of these difficulties result from additional complexities involving intertask communication and memory or CPU contention. Several of these topics will be examined with an eye towards techniques that can be applied to reduce overhead and improve performance of multi-task applications.

9.6.1 Intertask Communication

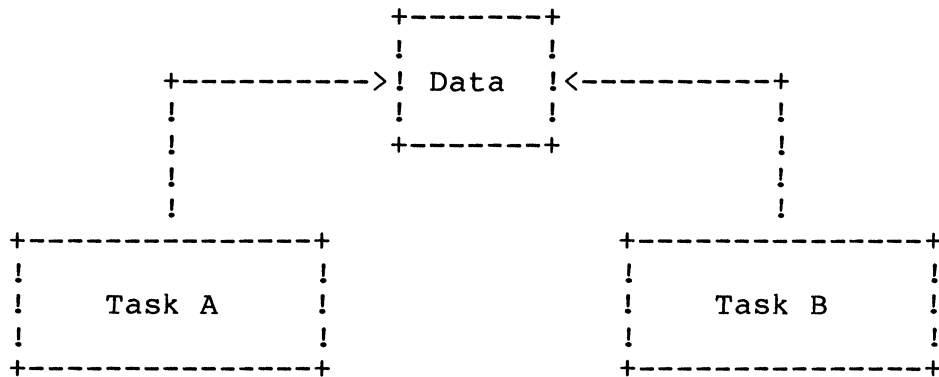
Very few, if any, multi-task applications consist of tasks that are totally independent of one another. Usually, the tasks cooperate in the completion of operations for the user of the application. By the very nature of cooperation it is imperative that the tasks be able to communicate quickly and efficiently. There are many methods of doing this. They range from merely sharing event flags to passing small amounts of data to passing entire regions of data between tasks. A subsequent section will deal with the subject of synchronizing tasks via event flags.

One common approach to intertask communication is to send data or send regions of data between tasks. If this is done incorrectly, performance problems are often the result. Let's look at one specific example. Say that a task cannot spare the two APRs necessary to map some cluster libraries but can spare one APR. The application is then constructed of two cooperating tasks that share data. One task is the main task which generates requests for the other task to fulfill. One possibility is to send data from one task to another using a send by reference (SREF\$) directive for every request. This could prove to be very time consuming as the system needs to process the SREF\$ and RREF\$ for every request.

MULTI-TASK APPLICATIONS

An alternative approach is to issue only one SREF\$/RREF\$ pair to set up a common region between the tasks. The tasks then communicate via this region and synchronize with event flags. All buffers of data that are necessary to be passed are kept in this region rather than being copied into the region whenever necessary. The final memory configuration of the two tasks is shown in the following diagram.

Tasks Sharing a Region



Another possibility, very similar to the one just presented is to use a read-write PLAS region or static common rather than a single SREF\$/RREF\$ pair. This accomplishes the same results and in essentially the same manner.

9.6.2 Memory usage

There are some simple things to avoid that may help reduce memory contention and the possibility of deadlock. If the tasks use cluster libraries, the NULLIB library may be used to reduce the amount of physical memory that must be occupied while executing in the task region.

9.6.3 Significant event impact

If two cooperating tasks within an application are using global event flags to synchronize themselves to events, then the proper method of setting a flag for the other task to react to is to declare a significant event after setting the flag. The following is a skeletal representation of the two cooperating tasks setting flags without the use of the significant event mechanism.

MULTI-TASK APPLICATIONS

Task	T1	Task	T2
SETF\$\$	#34	WTSE\$\$	#34
WTSE\$\$	#33	CLEF\$\$	#34
CLEF\$\$	#33	(code here)	
		SETF\$\$	#33

And below is a skeletal representation of the two cooperating tasks setting flags with the use of the significant event mechanism. Note that the only difference in the tasks is the declaration of a significant event after a flag has been set. This simple change provides a large amount of performance improvement.

Task	T1	Task	T2
SETF\$\$	#34	WTSE\$\$	#34
DECL\$\$		CLEF\$\$	#34
WTSE\$\$	#33	(code here)	
CLEF\$\$	#33	SETF\$\$	#33
		DECL\$\$	

9.6.4 Contention

If an application has multiple active tasks care should be taken to avoid contention problems, both memory and CPU contention. If it is not necessary for a task to run concurrently with other tasks but is necessary that it be available to perform some processing, then use should be made of the distinction between stopping and waiting. With directives that wait, the task is still competing for memory resources and could possibly create contention problems. Directives that cause the task to be stopped will remove the task from contention for memory and allow easier access for any other tasks that must still perform processing. In addition, tasks that stop may be checkpointed and remain in the checkpoint file until the stop bit is cleared. They will not contend with other tasks to be brought back into memory. To avoid possible memory fragmentation, it is suggested that, if possible, the task not be mapped to any commons while stopped.

Specifically, assume that you have a server task that is notified of an operation by an event flag. Rather than using WTSE\$, use STSE\$. Your task will become unstopped and start executing and competing for memory again. If it has been checkpointed it will be brought back into memory. Once the task has completed its function it can stop itself once again via the STSE\$ directive.

CONCURRENT APPLICATIONS

9.7 CONCURRENT APPLICATIONS

Concurrent applications refers to the ability to support two or more simultaneous applications. At the present time P/OS supports multi-task applications but not concurrent applications. P/OS maintains much information on a system wide basis, including logicals such as APPL\$DIR, APPL\$HELP and APPL\$MENU, and default directory. There also exists the possibility of task naming conflicts, event flag conflicts and even data file conflicts. For these and other reasons, P/OS does not currently support concurrent applications.

9.7.1 Only for "bounded" systems

Even though P/OS does not support concurrent applications, there are a number of applications that are being designed and written with the idea of becoming background applications. This may work in a very bounded environment. A bounded environment is one in which the machine and its workload are carefully tailored and in which help is available to solve contention problems and conflicts. Essentially, a bounded environment is one in which every machine may obtain individual attention or consideration from the application developer. A background application task is made known to the system by the /NOREMOVE switch in the install file or by the "NOREMOVE" request bit in a PROTSK (POSSUM) install call. "NOREMOVE" tasks and commons will be neither aborted nor removed when the application exits.

9.7.2 Exercise extreme care

Should you undertake the development of a background application, remember a few facts. At this time concurrent applications are not supported by P/OS. When a background application starts it should gather all the necessary information at that time. It should not rely on system wide information after it has been started, nor should it change such information. For a background task to rely, for example, on the default directory or the logical name APPL\$DIR would be disastrous for the background application. For the background application to change system wide information would be disastrous for the foreground application as well as confusing for the user. A background application should not use task or region names that might be used by a foreground application. This is essentially asking the background application to do the impossible, that is, to guess at a potential name of a future foreground application.

POOL CONSIDERATIONS

9.8 POOL CONSIDERATIONS

The system's dynamic storage region, or pool, can be another source of possible performance problems. It is in the best interest of applications to be careful with those functions that may result in pool depletion.

It is quite possible for an application to cause the system to exhaust all available free space in pool while attempting to service the requests generated by the application. The following sections examine circumstances which can lead to pool exhaustion.

9.8.1 Offspring Control Blocks

Offspring control blocks are one source of packets that build up and exhaust pool. When a parent task exits, the OCB's of its offspring are NOT deallocated. The OCB's are deallocated only when the child exits or emits status (EMST\$).

If you request (RQST\$ or RPOIS\$ without RP.OAL) a task rather than spawning it, there will be no OCB created.

NOTE

Prior to V2.0 of P/OS, if any task without an OCB either aborted or exited with outstanding I/O, the system would bugcheck. Even an OCB which no longer pointed to a parent (the parent exited before the offspring) would be sufficient to prevent this bugcheck. With V2.0, this type of bugcheck is limited to background (NOREMOVE) tasks.

9.8.2 Lock Blocks

If your application uses QIO's for file access rather than RMS then you should exercise great care. For example, say that your application uses RMS to open a file but uses QIO's to read and write blocks in that file. One possible side effect is that the improper setting of the SHR field in the RMS FAB block may cause a large number of record lock blocks to be generated. It may take a while to deplete system pool, but when it happens it will generate some very confusing messages for the user of the application or possibly the developer of the application. Care should be exercised when selecting the values to be placed in the SHR field.

POOL CONSIDERATIONS

If RMS is used for all file accesses, then RMS will manage all the record lock blocks and system pool will not be unnecessarily depleted.

9.8.3 Open Files

Any open file requires that the system use some amount of pool space. If you have a large number of files open you may be using a large amount of pool. If it is possible, reduce the number of files that are open at any one time. This will reduce pool requirements. Any files open when your task aborts could be damaged or left in strange states.

9.8.4 Attachment Descriptor Blocks

Attachment descriptor blocks may build up in pool if the task is not careful to specify that they should not be created unless necessary. For instance, use WS.NAT in the window block for a RREF\$ directive. This tells the directive that the attachment descriptor block should only be created if necessary.

9.8.5 Send Data Packets

Packets from the send data directive will pile up in secondary pool if the receiver is not receiving. If a large number of send data packets are issued and the receiver for some reason is not receiving any of them or even receiving them at a rate slower than they are being sent, then secondary pool will fill with the packets.

9.8.6 I/O Packets

It is very easy to fill up pool with I/O packets. All it takes is the use of the QIO\$ directive and the lack of any code to check for the eventual completion of the directive. One easy way to cause this to happen is to generate a large number of QIO\$'s to a relatively slow device. For instance, a large number of single character QIO\$'s to the line printer port (TT2:) would cause pool to fill up with I/O packets.

APPENDIX A

GLOSSARY

Application Developer

A person or persons using the Tool Kit to develop application programs for the Professional 300 series of personal computers.

Application Diskette Builder (ADB)

The Tool Kit software component that builds a distribution diskette for an application by copying the appropriate files from disk. The program runs on the Professional.

Application

The end result of the Tool Kit development cycle, a computer program that performs some useful service. In this manual, the term "application" is often a program and its related files.

Checkpointing

The process by which the Executive makes memory space and processor time available to higher-priority tasks by temporarily removing lower-priority tasks from memory and storing them on disk. Also called "rolling out".

End User

The person who will ultimately install and run your application.

Frame Development Tool (FDT)

A utility (editor) that creates menus, on-line help, and messages through an interactive session using forms. The displays are called frames, and frames are stored in frame definition files.

GLOSSARY

Host

An operating system that supports the Tool Kit as a layered product. VAX/VMS and RSX-11M/M-PLUS are Tool Kit host systems.

Interactive Program

A P/OS application that requires user interaction. On P/OS Hard Disk systems, only one interactive program can run at one time. An interactive program, however, can run at the same time as one or more noninteractive programs.

Noninteractive Program

A program that runs primarily without user interaction, such as Print Services and File Transfer. On P/OS Hard Disk systems, multiple noninteractive programs can run concurrently with an interactive program.

Object Module

A file containing a task in the form of unrelocated binary instructions and data.

P/OS

The Professional 300 series operating system. P/OS is a single-user, real-time, multitasking system based on RSX-11M-PLUS.

P/OS Executive

The "nucleus" of the P/OS Operating System.

PRO/FMS-11 (Forms Management System)

A tool that performs screen management and data input via predefined forms.

PRO/RMS-11 (Record Management System)

A tools that provides an interface to the P/OS file system.

PRO/SORT

A tool that reorders data files according to control or key fields within the input data records.

GLOSSARY

RSX-11M-PLUS

A real-time, multitasking, operating system that makes use of the enhanced hardware features and memory of PDP-11/44 and PDP-11/70 processors.

Task

The fundamental executable program unit.

Task Builder

A tool (sometimes called a "linker") that converts an object module into a task image by relocating code and data and resolving external references.

Task Image

A file that contains a loadable task in the form of relocated binary instructions and data.

Terminal Emulator

An application that allows the Professional to function as a terminal for the purpose of working on host systems.

Tool Kit

A set of software tools and documentation used to develop applications for the Professional 300 series of personal computers.

User Interface

The method by which an end user interacts with a task. The P/OS user interface consists of menus, on-line help, a message system, and standard use of function keys.

Workstation

A Professional 350 personal computer used in combination with a host system to perform a particular operation.

APPENDIX B

SPACE REQUIREMENTS FOR P/OS DISKETTE V1.7

Filename	Directory	File Size (in blocks)	Memory Size (in Kb)	Must be included for:
BASIC2.ERR	001002	8		Tool Kit BASIC-PLUS-2
C81DBG.HLP	001002	13		Tool Kit COBOL-81
C81LIB.TSK	ZZSYS	34	8	Tool Kit COBOL-81
C81RTE.TXT	001002	10		Tool Kit COBOL-81
CET.TSK	ZZSYS	123	56	Callable Editor
CGLFPU.TSK	ZZSYS	22	13	OPTIONS Graphics (CGL)
CMAIN.TSK	ZZSYS	86		All applications
COMLIB.TSK	ZZSYS	13	3	PRO/Communications
COMSETUP.DAT	ZZSYS	1		PRO/Communications
CPRNT.TSK	ZZSYS	33		Print services
CREDEL.TSK	ZZSYS	1		PRODIR
CTEX.MSG	ZZSYS	3		All applications
DBLRES.TSK	ZZSYS	35	15	Tool Kit DIBOL
DIBOLERR.MSG	001002	5		Tool Kit DIBOL
FMSERR.MSG	001002	2		PRO/FMS
GDSCOM.TSK	ZZSYS	1	20	OPTIONS Graphics
PASERR.MSG	001002	7		Tool Kit PASCAL
PASRES.TSK	ZZSYS	34	16	Tool Kit PASCAL, PRO/SORT, Callable Editor
PBFSML.TSK	ZZSYS	34	16	Tool Kit BASIC-PLUS-2
POSRES.TSK	ZZSYS	38	18	All applications
PRINT.MSG	ZZSYS	6		Print services
PROF77.MSG	001002	7		Tool Kit FORTRAN-77
PROF77.TSK	ZZSYS	34	16	Tool Kit FORTRAN-77
PROSE.HLP	ZZSYS	66		Callable Editor
PROSE.MNU	ZZSYS	3		Callable Editor

PROSE.MSG	ZZSYS	7		Callable Editor
PROSE.UDK	ZZSYS	10		Callable Editor
PROSORT.SYS	ZZSYS	26		PRO/SORT
PROSORT.TSK	ZZSYS	74	42	PRO/SORT
RMSERROR.MSG	ZZSYS	3		POSRES
SUMFBI.TSK	ZZSYS	1		PROFBI
SYSTEM.MNU	ZZSYS	4		OLDFIL/NEWFIL
XKDRV.TSK	ZZSYS	16		PRO/Communications
XTDRV.TSK	ZZSYS	17	6	TMS

APPENDIX C

POSRES STATUS BLOCK CODES

POSRES uses the status block parameter to return error and status information to the calling program. It is recommended that your task check the status block after each POSRES call.

Decimal status values are returned to the calling task in a two-word integer array. The first column of Table C-1 shows the values returned in the first word of the status array. The second column lists the values returned in the second word of the status array, except for menu routine errors, which are shown in Table C-2.

In Table C-1, the numbers one and two represent the first and second status block words, respectively. In your application, the first word may be array element zero, one, or n, depending on which programming language you are using. For example, BASIC-PLUS-2 numbers arrays from zero while PASCAL lets you define your own numbering scheme.

Table C-1: POSRES Status Values

<u>Status Block Words</u>		<u>Description</u>
<u>1</u>	<u>2</u>	
+1	1 through 12	For menus, option selection was successful and the second word contains the ordinal option number.
	ASCII code	For GETKEY, the second word contains an ASCII decimal code representing a keyboard key.
	Undefined	For other routines, there was no error.
+2	0	A record or field was truncated.

POSRES STATUS BLOCK CODES

Table C-1 (cont.)

<u>1</u>	<u>2</u>	<u>Description</u>
	-6	A message sent to the message board was truncated to 59 or fewer characters and displayed.
	Key code	For GETKEY, the second word contains one of the function key codes listed in Appendix D.
	Key value	For PRSCSI routine, a valid CSI sequence was entered.
-1	DSW	RSX DSW error. See the <u>P/OS System Reference Manual</u> for error codes.
-2	I/O code	I/O status error code returned from a QIO\$ directive. See the <u>P/OS System Reference Manual</u> .
-3		File access error.
	-1	Index record not 256 bytes long
	-2	No match during index operation
	-3	File index record is greater than one block
	-4	File is not open
	-5	Frameid is not in Radix-50 character set
-4	See Table C-2	Error executing help routine.
-5	See Table C-2	Error executing menu routine.
-6	See Table C-2	Error executing dynamic menu routine.
-7	-11	Invalid CSI sequence found
	-12	No CSI sequence found
	-13	File extension error
-9		Calling parameter error
-10	0	Insufficient buffer space
-11		Short message error
	-1	No matching entry number
-12	RMS error code	PRO/RMS-11 File access error. See the PRO/RMS-11 manual set for error codes.
-13	See Table C-2	Error executing menu unpack routine.

POSRES STATUS BLOCK CODES

Table C-1 (cont.)

<u>1</u>	<u>2</u>	<u>Description</u>
-14	Key code	Option selection failed. The second word contains one of the function key codes listed in Appendix D. For example, the value 14 indicates that the user pressed the ADDTNL OPTIONS key.
-15	See Table C-2	Error displaying frame specified with frameid.
-16	See Table C-2	Error executing multiple-choice menu routine.
-17	See Table C-2	Error executing menu pack or unpack routine.
-18		Error executing OLDFIL routine.
	-1	No choices made
	-2	No files found
	-3	Error in wildcard selection
-19		Buffer error. The buffers FL\$BUF and MM\$BUF are not large enough.
-20	-1	Message rejected. The Message/Status Display is full. The Display lists up to 255 messages.

Table C-2: Menu Service Routine Errors

<u>Word Two</u>	<u>Description of Error</u>
-1	Option number greater than maximum
-2	Multiple once-only fields
-4	Error in keyword definition
-5	Title or text field length error
-6	Text field length error
-8	Argument error or unknown fieldid
-9	Buffer error
-10	Text or option line number greater than maximum
-11	Error in menu packing

POSRES STATUS BLOCK CODES

Table C-2 (cont.)

<u>Word Two</u>	<u>Description of Error</u>
-12	No options for multiple-choice menu
-13	Multiple-choice menu limits responses to zero
-14	More responses allowed than options
-15	No help available

APPENDIX D

FUNCTION KEY NAMES AND CODES

<u>Code</u>	<u>PRO Label</u>	<u>P/OS Label</u>
1	F1	F1
2	F2	F2
3	F3	BREAK
4	F4	SETUP
5	F5	F5
6	F6	Reserved
7	F7	RESUME
8	F8	CANCEL
9	F9	MAIN SCREEN
10	F10	EXIT
11	F11	F11
12	F12	F12
13	F13	F13
14	F14	ADDTNL OPTIONS
15	HELP	HELP
16	DO	DO
17	F17	F17
18	F18	F18

FUNCTION KEY NAMES AND CODES

<u>Code</u>	<u>PRO Label</u>	<u>P/OS Label</u>
19	F19	F19
20	F20	F20
21	FIND	FIND
22	INSERT HERE	INSERT HERE
23	REMOVE	REMOVE
24	SELECT	SELECT
25	PREV SCREEN	PREV SCREEN
26	NEXT SCREEN	NEXT SCREEN
27	up arrow	up arrow
28	left arrow	left arrow
29	down arrow	down arrow
30	right arrow	right arrow
31	PF1	PF1
32	PF2	PF2
33	PF3	PF3
34	PF4	PF4
(Application Mode only)		
35	minus	minus
36	comma	comma
37	period	period
38	ENTER	ENTER
39	0	0
40	1	1
41	2	2
42	3	3

FUNCTION KEY NAMES AND CODES

<u>Code</u>	<u>PRO</u> <u>Label</u>	<u>P/OS</u> <u>Label</u>
43	4	4
44	5	5
45	6	6
46	7	7
47	8	8
48	9	9

APPENDIX E
DOCUMENTATION DIRECTORY

This appendix lists the documentation provided with the Professional Host Tool Kit and PRO/Tool Kit.

E.1 VOLUME 1: INTRODUCTION

Host Tool Kit Installation Guide and Release Notes	AA-N616C-TK *
Tool Kit User's Guide	AA-N617D-TK
Tool Kit Reference Manual	AA-BT74A-TH
FMS-11/RSX Release Notes	AA-H857A-TC
PRO/FMS-11 Documentation Supplement	AA-P103B-TK
FMS-11/RSX Software Reference Manual	AA-H855A-TC

E.2 VOLUME 2: VIDEO

Terminal Subsystem Manual	AA-N623B-TK
CORE Graphics Library Manual	AA-N619B-TK
(Update 1)	AD-N619B-T1
PRO/GIDIS Manual	AA-Y660A-TK
(Update 1)	AD-Y660A-T1

E.3 VOLUME 3: TASK BUILDING

RSX-11M/M-PLUS Task Builder Manual	AA-L680B-TC
------------------------------------	-------------

* Provided with the PRO/Tool Kit but intended for use only with the Host Tool Kit.

VOLUME 4: P/OS SYSTEM

E.4 VOLUME 4: P/OS SYSTEM

P/OS System Reference Manual
(Update 1)
(Update 2)

AA-N620A-TK
AD-N620A-T1
AD-N620A-T2

E.5 VOLUME 5: PRO/RMS-11

PRO/RMS-11: An Introduction
PRO/RMS-11 Macro Programmer's Guide
RSX-11M/M-PLUS RMS-11 User's Guide

AA-P098A-TK
AA-P099A-TK
AA-L669A-TC

E.6 VOLUME 6: MACRO PROGRAM DEVELOPMENT

IAS/RSX-11 ODT Reference Manual
IAS/RSX-11 ODT Supplement
PDP-11 MACRO-11 Language Reference Manual
Guide to Writing a P/OS I/O Device Driver
and Advanced Programmer's Notes

AA-M507A-TC
AA-P193B-TK
AA-V027A-TC
AA-BT73A-TH

E.7 VOLUME 7: PRO/DECNET

PRO/DECnet Tool Kit Release Notes
Introduction to DECnet
PRO/DECnet Tool Kit Installation Guide
PRO/DECnet Programmer's Reference Manual

AA-AV71A-TK
AA-J055C-TK
AA-AV70A-TK
AA-AV69A-TK

E.8 PRO/TOOL KIT VOLUME

PRO/Tool Kit Installation Guide and Release Notes
Command Language and Utilities Manual
RSX-11M/M-PLUS RMS-11 Utilities Manual

AA-X911B-TH
AA-X912A-TH
AA-L670A-TC

INDEX

-A-

Action string
 description, 8-8
 global, 8-8
 option, 8-8
ADB
 see Application Diskette
 Builder
Additional Options, 8-7, 8-10,
 8-12, 8-13
APPL\$DIR, 1-4, 6-11, 8-10, 8-19,
 9-14
APPL\$HELP, 9-14
APPL\$MENU, 9-14
Application directory, 4-3, 4-4,
 8-28
 description, 4-1
 P/OS Diskette, 6-10
 P/OS Hard Disk, 6-11
Application Diskette Builder, 4-6,
 A-1
 description, 2-3
Asynchronous System Trap, 3-6,
 9-6, 9-10
Attachment Descriptor Block, 9-16

-B-

BASIC-PLUS-2, B-1
 array parameter, 3-8
 description, 2-1
 external subprogram call, 3-8
 string parameter, 3-8
BIGVOLUME, 6-4
Bitmap
 documentation, 1-10

-C-

Callable Editor Task, 2-8, B-1,
 B-2
Callable Sort Task, 2-8
Calling Sequence Convention,
 PDP-11 R5, 3-7
CET
 see Callable Editor Task

CGL

 see CORE Graphics Library
Character set
 documentation, 1-7
Checkpointing, 5-1, 5-4, A-1
CLOG\$ system directive, 6-10, 9-2
Cluster library, 5-3
 default, 7-4
 description, 3-6
COBOL-81, B-1
 description, 2-1
 external routine call, 3-8
COMLIB, 2-4, 3-7
Communications, 3-4, 3-7
Communications Services
 description, 2-3
Console cable
 use of in debugging, 4-5
Convention, PDP-11 R5 Calling
 Sequence, 3-7
CORE Graphics Library, 2-4, 3-5,
 3-7, 4-2, 7-3, B-1
 documentation, 1-7
CRAW\$ system directive, 9-2
CSI sequence
 parsing, 8-25
Cursor
 turning off, 9-7

-D-

DCL

 see Digital Command Language
DEC Multinational Character Set
 documentation, 1-7
DECnet
 description, 2-5
 documentation, 1-10
Device Driver
 documentation, 1-9
Device name
 default, 6-3
 setting, 6-4
 description, 6-2
DIBOL, B-1
 description, 2-2
 external subroutine call, 3-9

INDEX

Digital Command Language, 1-4,
6-10, 6-11
 CREATE command, 4-1
 documentation, 1-11
 EDIT command, 4-2
 INSTALL command, 4-3
 LINK command, 7-1
 RUN command, 4-4
 SET command, 4-3, 6-4, 6-5
 SPAWN command, 4-2
Directory name
 default, 9-2
 setting, 6-5
 description, 6-4
Disk/Diskette Services, 2-6, 3-4,
3-5, 4-1, 6-4, 6-5, 6-11, 8-4
DLOG\$ system directive, 6-10, 9-2
Dynamic menu
 description, 8-9, 8-10
 displaying, 8-11
 frame pointer, 8-19
 from static menu, 8-11

-E-

EDT, 4-2
Eight-bit characters
 using, 1-5, 9-6, 9-9
ELAW\$ system directive, 9-2
EMST\$ system directive, 9-15
Equivalence name
 definition, 6-7
Event flag
 use of, 8-26
EXTK\$ system directive, 9-1

-F-

Fast Install, 4-4, 6-11
 description, 2-6
Fatal error
 handling, 8-22
FCS-11
 see File Control Services
FDT
 see Frame Development Tool
File
 pre-allocation, 9-3
 pre-extension, 9-3
 use of identifier, 9-3
 when to open, 9-3

File Control Services
 description, 2-6
File name
 description, 6-5
File Selection Menu, 8-13
File Services, 3-4, 6-4, 6-5, 8-4,
8-6
File specification
 description, 6-1
File Transfer utility, 4-4
File type
 description, 6-6
Files-11
 use of logical names, 6-9
FIRSTAPPL.PTR, 1-4
FMS-11
 see Forms Management System
Forms Management System, 4-2, 4-3,
7-3, B-1
 description, 2-6
 documentation, 1-6
FORTRAN-77, B-1
 description, 2-2
 external subroutine call, 3-9
Frame Development Tool, 4-2, 8-4,
8-10, 8-19, 8-21, 8-30, A-1
 CONVERT command, 4-3
 description, 2-7
Frame pointer
 description, 8-19
 use of, 8-20
Function key, 8-2
 codes, D-1
 description, 8-22
 menu option selection, 8-7
 menu programming, 8-12
 programming, 8-24
 use of, 8-23, 9-9

-G-

GIDIS, 3-5
 description, 2-5
 documentation, 1-7

-H-

Help
 programming
 see POSRES

INDEX

Help file
 opening, 8-19
Help frame
 default
 specifying, 8-19
Help menu
 description, 8-14
 key processing, 8-15
Help structure, 8-2
 activating, 8-20
 designing, 8-14
 implementing, 8-19
Help text frame
 description, 8-16
 half-screen, 8-16
 key processing, 8-16
Host Tool Kit
 description, 1-2
 documentation, 1-6
 using, 1-5

-I-

Installation
 ASSIGN HELP command, 6-11, 8-12,
 8-14, 8-19, 8-20
 ASSIGN MENU command, 6-11, 8-10,
 8-12, 8-14
 command file, 2-3, 4-1, 4-2,
 4-4, 6-11

-K-

Keyboard, 8-28
 description, 3-1
 documentation, 1-7
 label strip, 8-23
Keystroke
 input routine, 8-24
Keyword, 8-5, 8-8

-L-

Lock block, 9-15
Logical name
 description, 6-7
 Files-11 use, 6-9
 manipulating, 6-10
 RMS-11 translation, 6-9
 table, 6-7

Logical unit number
 see LUN
LUN
 use of, 8-26

-M-

MACRO-11
 see Professional Macro
 Assembler
MAP\$ system directive, 9-2
Memory
 physical, 5-4
 virtual, 5-2
Memory management, 9-2
Menu
 key processing, 8-7
 programming
 see POSRES
 user perception, 8-2
Message
 use of, 8-2
Message file
 description, 8-21
Message frame
 reading, 8-22
Message/Status Display, 8-2, 8-22,
 C-2, C-3
Multiple-choice menu, 8-4, 8-6
 displaying, 8-11
 option selection, 8-6
 unpacking, 8-10
 use of, 8-7

-N-

New File Specification form, 8-12
NULLIB, 7-5, 9-12

-O-

ODT
 see On-Line Debugging Tool
Offspring Control Block, 9-15
On-Line Debugging Tool
 description, 2-7
 documentation, 1-9
Overlay Descriptor Language, 7-4,
 7-6 to 7-7
 .FCTR directive, 7-6
 .ROOT directive, 7-6

INDEX

Overlaying, 5-2

-P-

PAB

see Professional Application
Builder

Parameter

data type checking, 3-7
position of, 3-7

PASCAL, B-1

description, 2-3
external procedure call, 3-9
external procedure declaration,
3-9

READONLY attribute, 3-10

UNSAFE attribute, 3-10

PDP-11 R5 Calling Sequence
Convention, 3-7

Physical memory, 5-4

PLAS directive, 9-1, 9-10

PMA

see Professional Macro
Assembler

P/OS

access from high-level
languages, 3-7

components, 3-4

configurations, 5-1

current device, 8-27, 8-28

description, 1-1

Diskette

description, 3-2

Executive

description, 3-4

documentation, 1-8

file specification, 6-1

Hard Disk

description, 3-2

overview, 3-1

user interface, 3-3

POSRES, 3-7, 4-2, 6-11, 7-3, 8-1
to 8-32, B-2

and FDT, 2-7, 8-4

buffer names, 8-29

clearing buffers, 8-11

description, 2-7, 8-4

DMENU routine, 8-11, 8-27, 8-29

DPACK routine, 8-10, 8-29

FATLER routine, 8-22, 8-27

POSRES (Cont.)

GETKEY routine, 8-24, 8-27, C-1,
C-2

global symbols, 8-26

HCLOSE routine, 8-27, 8-29

HELP routine, 8-20, 8-27, 8-29

HFILE routine, 8-13, 8-14, 8-19,
8-20, 8-27, 8-29

HFRAME routine, 8-13, 8-14,
8-20, 8-27, 8-29

MCLOSE routine, 8-12, 8-27,
8-29

menu programming, 8-11

MENU routine, 8-10, 8-27, 8-29

MFILE routine, 8-10, 8-12, 8-14,
8-27, 8-29

MFRAME routine, 8-10, 8-12,
8-14, 8-27, 8-29

MMENU routine, 8-11, 8-27, 8-29

MPACK routine, 8-10, 8-29

MSGBRD routine, 8-22, 8-26,
8-27

MUNPK routine, 8-11, 8-29

NEWFIL routine, 8-12, 8-27,
8-29, B-2

OLDFIL routine, 8-10, 8-13,
8-27, 8-29, B-2, C-3

overlaying buffers, 8-31

PRSCSI routine, 8-25, 8-27, C-2

RDMSG routine, 7-3, 8-22, 8-27

reducing buffer sizes, 8-29

status block, 8-11, 8-24

status codes, C-1

suggested maximum buffer sizes, (
8-29

task image requirements, 8-25

use of buffers, 8-9

use of by languages, 8-25

WTRES routine, 7-3, 8-25, 8-27

POSSUM, 3-7, 7-3, 7-5

description, 2-8

PRODIR routine, 6-5

PROFBI routine, B-2

PROLOG routine, 6-4, 6-5, 6-10,
9-2

PROTSK routine, 9-14

Print Services, 3-4, 3-5, 5-2,
B-1

description, 2-7

Printer

use of in debugging, 4-5

INDEX

PRO/Communications, 2-4, B-1, B-2
 PRO/DECnet
 see DECnet
 PRO/Dispatcher, 3-5
 PRO/FMS-11
 see Forms Management System
 PRO/GIDIS
 see GIDIS
 PRO/RMS-11
 see Record Management Services
 PRO/SORT, B-1, B-2
 description, 2-8
 PRO/Tool Kit
 application directory, 1-4
 description, 1-2
 documentation, 1-6, 1-11
 startup command procedure, 1-4
 using, 1-4
 Professional
 hardware configurations, 1-2,
 3-1
 Professional Application Builder,
 4-2, 4-3, 7-1 to 7-7, 8-22,
 8-25
 ASG option, 8-27
 CLSTR option, 7-4
 command file, 7-3
 CP switch, 7-4
 description, 2-3
 documentation, 1-8
 EXTSCT option, 8-28, 8-29
 FP switch, 7-4
 GBLDEF option, 8-26, 8-27
 MA switch, 7-4
 PRO/Tool Kit, 7-1
 RSX-11M/M-PLUS, 7-2
 SP switch, 7-4
 UNITS option, 8-26
 VAX/VMS, 7-1
 Professional Macro Assembler, 2-3,
 2-5, 2-7, 2-8
 description, 2-2
 documentation, 1-9
 P/OS routine call, 3-10
 PROSE, 3-4, 4-2
 description, 2-8
 Pseudo-device, 6-3

-Q-

QIO system directive, 9-6

QIO\$ system directive, 9-7, 9-10,
 9-16
 QIOW\$ system directive, 9-7

-R-

R5 Calling Sequence Convention,
 PDP-11, 3-7
 Record Management Services, 2-6,
 3-5, 3-7, 6-7, 7-3, 9-15
 description, 2-8
 documentation, 1-8
 input parsing, 9-2
 logical name translation, 6-9,
 6-10, 6-11, 9-2
 utilities
 documentation, 1-11
 RMS-11
 see Record Management Services
 RMSRES, 7-5
 RPOI\$ system directive, 9-15
 RQST\$ system directive, 9-15
 RREF\$ system directive, 9-11,
 9-16
 RSX-11M/M-PLUS
 description, 1-1

-S-

Sequence Convention, PDP-11 R5
 Calling, 3-7
 Single-choice menu, 8-4, 8-5
 option selection, 8-5
 packing, 8-10
 static, 8-10
 use of, 8-6
 SREF\$ system directive, 9-10,
 9-11
 STARTUP.COM, 1-4
 Static menu
 description, 8-9
 frame pointer, 8-19
 Storage media
 description, 3-1
 STSE\$ system directive, 9-13
 SYSDISK, 6-11
 System Services
 see POSSUM
 System Task Directory, 3-6
 System unit
 description, 3-1

INDEX

-T-

Task Builder
 see Professional Application
 Builder
 documentation, 1-8
Telephone Management System, 2-4,
 B-2
Terminal
 use of in debugging, 4-5
Terminal Emulator, 2-4, 3-4, 4-4,
 A-3
 documentation, 1-7
Terminal subsystem, 3-5
 documentation, 1-7
TLOG\$ system directive, 6-10, 9-2
TMS
 see Telephone Management System,
 2-4

-U-

UMAP\$ system directive, 9-2
User Interface Services
 see POSRES

USERDISK, 6-4
USERFILES, 6-5

-V-

Version number
 description, 6-6
Video bitmap
 documentation, 1-10
Video monitor, 8-28
 description, 3-1
 documentation, 1-7
 in debugging, 4-5
Virtual Address Space, 5-2

-W-

Wild-card, 6-7
Wild-card specification, 8-13
WTSE\$ system directive, 9-13

-X-

XKDRV, 2-4

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized?
Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____

or
Country

Please cut along this line

----- Do Not Tear - Fold Here and Tape -----

digital

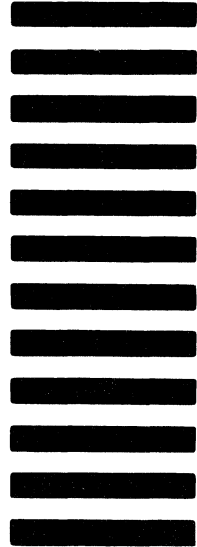


No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754



----- Do Not Tear - Fold Here -----