94-003 1668127

Micro/RSX Guide to Advanced Programming

Order No. AA-AB43C-TC

Micro/RSX Version 4.0

Digital Equipment Corporation Maynard, Massachusetts

First Printing, December 1983
Revised, July 1985
Revised, September 1987

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	EduSystem	UNIBUS
DEC/CMS	IAS	VAX
DEC/MMS	MASSBUS	VAXcluster
DECnet	MicroPDP–11	VMS
DECsystem–10	Micro/RSX	VT
DECSYSTEM–20	PDP	
DECUS	PDT	
DECwriter	RSTS	digital
DIBOL	RSX	

ZK3069

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	CANADA	INTERNATIONAL
Digital Equipment Corporation	Digital Equipment of Canada Ltd.	Digital Equipment Corporation PSG Business Manager
P.O Box CS2008	100 Herzberg Road	c/o Digital's local subsidiary
Nashua, New Hampshire 03061	Kanata, Ontario K2K 2A6	or approved distributor
	Attn: Direct Order Desk	

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a trademark of the American Mathematical Society.

Contents

Preface ix

Summary of Technical Changes xiii

Chapter 1 The Program Development Environment

1.1 Software Tools 1-1
1.1.1 DIGITAL Command Language 1-3
1.1.2 EDT Text Editor 1-3
1.1.3 PDP-11 MACRO-11 1-3
1.1.3.1 Source Input 1-4
1.1.3.2 Source File Directives 1-4
1.1.3.3 How the Assembler Works 1-5
1.1.4 Task Creation 1-5
1.1.5 Debugging Aids 1-6
1.1.5.1 On-Line Debugging Tool 1-6
1.1.5.2 Postmortem Dump 1-7
1.1.5.3 Snapshot Dump 1-7
1.1.6 General Utilities 1-7
1.1.6.1 Cross-Reference Processor 1-8
1.1.6.2 Library Operations 1-8
1.2 System Software 1-8
1.2.1 System Directives—Macro Libraries 1-8
1.2.2 System Subroutines—Object Libraries 1-9
1.3 Hardware 1-10
1.3.1 Disks 1-11
1.3.2 Terminals 1-11
1.3.3 Printers 1-11
1.4 Overview of the Program Development Process 1-12

Chapter 2 Creating MACRO-11 Source Files

2.1 MACRO-11 Skeleton Source File Format 2-1
2.2 Creating a Source File from a Skeleton File 2-9
2.2.1 Performing the Initial Input 2-10
2.2.2 Inserting Blank Lines in Text 2-10
2.2.3 Exiting from EDT 2-10
2.2.4 Creating a Source File from the Skeleton 2-12

Chapter 3 Assembling and Correcting a Program Module

3.1 Performing a Diagnostic Run on a Source File 3-1
3.2 Errors Encountered During Assembly 3-2
3.2.1 MACRO-11 Error Code A 3-2
3.2.2 MACRO-11 Error Code U 3-3
3.2.3 MACRO-11 Error Code Q 3-3
3.2.4 MACRO-11 Error Code E 3-3
3.3 Generating a Program Module and a Listing 3-4
3.4 Examining a Listing at the Terminal 3-5
3.5 Generating a Cross-Reference Listing 3-5
3.6 Printing a Copy of Listings 3-6
3.7 Cleaning Up the Disk Directory 3-6

Chapter 4 Building and Testing a Task

4.1 Creating a Task Image 4-1
4.1.1 Supplying a Single Object Module 4-1
4.1.2 Supplying Multiple Object Modules 4-2
4.2 Task Builder Defaults 4-3
4.3 Generating a Map and a Global Cross-Reference Listing 4-3
4.3.1 Requesting a Map and a Global Cross-Reference Listing 4-4
4.3.2 Examining the Map at the Terminal 4-4
4.3.3 Requesting a Full Map 4-4
4.4 Running the Task and Correcting Typical Errors 4-5

Chapter 5 Using Debugging Aids

5.1 Using the On-Line Debugging Tool 5-1
5.1.1 Including ODT in a Task 5-2
5.1.2 Preparing to Use ODT 5-2
5.1.3 Setting Up the Task 5-2
5.1.4 Relocation Registers 5-2
5.1.5 Examining Locations 5-4
5.1.6 Setting Breakpoints Within the Task 5-6
5.1.7 Changing the Contents of Locations with ODT 5-7
5.1.8 Error Conditions and Terminating Task Execution 5-8
5.2 Using the Postmortem Dump 5-9
5.3 Using the Snapshot Dump 5-9

Chapter 6 Creating and Using Program Libraries

6.1 Creating and Using a Macro Source Library 6-1
6.1.1 Creating the Macro Library 6-1
6.1.2 Using the Macro Definitions from the Library 6-3
6.2 Creating and Using an Object Module Library 6-3
6.2.1 Creating the Object Module Library 6-3
6.2.2 Using the Object Modules from the Library 6-4
6.2.3 Using the Library to Resolve Undefined Global Symbols 6-5
6.2.4 Dual Use of the Library 6-6
6.3 Maintaining User Libraries 6-6
6.3.1 Adding Modules to a Library 6-6
6.3.2 Replacing a Module in a Library 6-7
6.3.3 Obtaining Information About a Library 6-7

Chapter 7 Using Logical Name Commands

7.1 Logical Names 7-1
7.1.1 Logical Name Tables 7-2
7.1.2 Displaying Logical Name Table Entries 7-2
7.1.3 How to Create and Delete Logical Names 7-3
7.1.4 Logical Name Translation 7-4
7.1.5 Iterative Translation 7-4
7.2 ASSIGN Command 7-5
7.3 DEASSIGN Command 7-9
7.4 DEFINE Command 7-12
7.5 SHOW ASSIGNMENTS and SHOW LOGICALS Commands 7-15

Chapter 8 Guidelines for Creating an Optional Software Package

8.1 Preparing to Create a Diskette Package 8-2
8.1.1 Determining Required Number of Diskettes 8-3
8.1.2 Grouping Optional Software Files 8-3
8.1.3 Setting Up Files 8-3
8.1.4 Creating the INSTALL.DAT File 8-3
8.1.4.1 INSTALL.DAT File Statements 8-4
8.1.4.2 INSTALL.DAT File Examples 8-5
8.1.5 Creating the INS File 8-6
8.1.5.1 INS File Statements 8-7
8.1.5.2 Conditionals in INS File Statements 8-12
8.1.5.3 INS File Examples 8-13
8.1.6 Creating a Diskette Kit 8-21
8.1.6.1 Copying Single Volume Files 8-22
8.1.6.2 Copying Multivolume Files 8-24
8.2 Preparing to Create a Tape Package 8-25
8.2.1 Creating the Template INS File 8-26
8.2.2 Optimizing Your Template INS File 8-27
8.2.2.1 Merging BACKUP_SET Statements 8-28
8.2.2.2 Separating BACKUP_SET Statements 8-28
8.2.3 Editing INSTALL.DAT and Template INS Files 8-29
8.2.4 Organizing the Optional Software Files 8-29
8.2.4.1 Organizing Files Using Diskettes 8-29
8.2.4.2 Organizing Files on the Fixed Disk 8-31
8.2.5 Creating the Tape Kit 8-31
8.2.5.1 Copying Files from Diskettes to Tape 8-31
8.2.5.2 Copying Files from Fixed Disk to Tape 8-33
8.3 Documenting the Installation Procedure 8-34
8.4 Error Messages 8-35

Appendix A The MACRO Command

A.1 MACRO A-1

Appendix B Prototype Installation Guide for a Diskette Kit

B.1 Introduction to Micro/RSX <PRODUCT_NAME> Installation B-1
B.2 Checking Required Software B-1
B.3 Installing the <PRODUCT_NAME> Software B-3
B.4 Correcting Possible Errors B-5
B.5 Logging in to Micro/RSX B-8

Appendix C Prototype Installation Guide for a Tape Kit

C.1 Introduction to Micro/RSX <PRODUCT_NAME> Installation C-1
C.2 Checking Required Software C-1
C.3 Installing the <PRODUCT_NAME> SOFTWARE C-2
C.4 Correcting Possible Errors C-4
C.5 Logging in to Micro/RSX C-6

Appendix D Micro/RSX Reference and Customization Files

Index

Examples

2–1 Sample Skeleton Source File 2-5
2–2 Creating the Skeleton File SKEL.MAC 2-11
2–3 Source Code for FILE.MAC 2-13
2–4 Source Code for FILEA.MAC 2-15
2–5 Source Code for FILEB.MAC 2-17
5–1 Memory Allocation Synopsis from Task BUG Map 5-3
5–2 Portion of Assembly Listing for NUMA 5-4
6–1 MACRO-11 Library Source Definitions 6-2
8–1 Using COPY=YES with One Diskette 8-5
8–2 Using COPY=NO with Two Diskettes 8-6
8–3 Using COPY=YES with Three Diskettes 8-6
8–4 Creating an Option on a Single Diskette 8-14
8–5 Creating an Option on Multiple Diskettes 8-15
8–6 Using Conditionals in an INS File 8-16
8–7 Using Conditionals and Multivolume Files 8-18
8–8 Inserting Modules into SYSLIB 8-20

Figures

1–1 The Program Development Process 1-13
2–1 MACRO-11 Source File Format 2-3
2–2 MACRO-11 Source Statement Format 2-4

Tables

1–1 DIGITAL-Supplied Macro Libraries 1-8
1–2 DIGITAL-Supplied Object Libraries 1-10
3–1 Terminal Output Control Commands 3-5
8–1 Angle Bracket Symbols 8-34
A–1 The Arguments to the /ENABLE and /DISABLE Qualifiers Assembly Functions Disabled by Default A-3
A–2 The Arguments to the /SHOW and /NOSHOW Qualifiers Listing Functions Disabled by Default A-5

Preface

Manual Objectives

The *Micro/RSX Guide to Advanced Programming* introduces you to the program development environment used on a Micro/RSX operating system. It provides a synopsis of the information immediately useful to get started in the program development process. This manual also includes guidelines for program design and for producing an optional software package.

Intended Audience

This manual is intended for the programmer who is already familiar with the basic operations of a Micro/RSX operating system: gaining access to the system, using the terminal and related devices, and requesting simple Executive services through the DIGITAL Command Language (DCL) command interface. The *Micro/RSX Guide to Advanced Programming* addresses assembly language programming because that language is the one provided with the Advanced Programmer's Kit. However, most of the topics covered for the assembly language programmer—using a text editor, creating an executable image, and using library facilities—apply to programmers using any computer language.

Structure of This Document

This manual is meant to be read as you use the system. For this reason, the examples are presented in an order that you can follow at the terminal. Rather than demonstrate the complexity of the system, these examples are designed to demonstrate practical program development operations.

Chapter 1 introduces the software and hardware on which you develop programs.

Chapter 2 describes how to create an assembly language source program using a skeleton file and text editor.

Chapter 3 describes how to use the PDP-11 MACRO-11 assembler to generate an object module.

Chapter 4 describes how to use the Task Builder (TKB) to link object modules to create a loadable task image.

Chapter 5 introduces debugging aids and discusses how to use them.

Chapter 6 describes how to create and maintain a library of macro source statements and a library of object module subroutines.

Chapter 7 describes how to use logical name commands.

Chapter 8 describes how to create an optional diskette and tape package.

Appendix A describes how to use the MACRO command to assemble a program.

Appendix B provides a prototype installation guide for documenting the installation procedure for an optional software diskette kit.

Appendix C provides a prototype installation guide for documenting the installation procedure for an optional software tape kit.

Appendix D describes the files used to customize a Micro/RSX operating system.

Associated Documents

Because the Micro/RSX operating system is a subset of the RSX-11M-PLUS operating system, most of the manuals provided with the Advanced Programmer's Kit are RSX-11M-PLUS operating system manuals. However, the information in these manuals also applies to Micro/RSX systems (differences are clearly distinguished and noted). Before using the documentation, please read the *Micro/RSX Release Notes* to orient yourself with the manuals provided in the Advanced Programmer's Kit. Refer to the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2* for the information you need to gain access to the services provided with the Micro/RSX operating system.

Refer to the *PDP-11 MACRO-11 Language Reference Manual* if you need more information on developing MACRO-11 programs.

Conventions Used in This Document

The following conventions are observed in this manual.

Convention	Meaning
xxx	A 1- to 3-character key symbol. For example, ESC indicates the ESCAPE key, RET indicates the RETURN key, and LF indicates the LINE FEED key.
CTRL/A	A symbol that indicates the CTRL key; it must be held down while another key is pressed. For example, CTRL/C means to hold down the CTRL key while typing C.
^	A circumflex represents the system response on some terminals to receiving a control character such as CTRL/A. For example, when you type CTRL/Z while running certain system tasks on some terminals, the system echoes ^Z.
"print" "type"	As these words are used in the text, the system prints and the user types.
$	The DIGITAL Command Language (DCL) prompting character.
red ink	Color of ink used to show all user-entered commands in examples.

Convention	Meaning
black ink	Base color of ink used throughout manual. In examples, however, black ink has a special meaning: All output lines and prompting characters that the system prints or displays are shown in black ink.
,	A comma separates parameters in commands.
.	A period separates the file name from the file type in a file specification.
;	A semicolon separates the file type from the file version number in a file specification.

Summary of Technical Changes

This revision of the *Micro/RSX Guide to Advanced Programming* contains technical features and documentation changes included in Micro/RSX operating system Version 4.0.

Information Moved to Other Manuals

The following chapters have been moved from the *Micro/RSX Guide to Advanced Programming* to the manuals listed:

- Loadable Crash Dump Support is now included in the *RSX-11M-PLUS and Micro/RSX Crash Dump Analyzer Reference Manual.*
- The Executive Debugging Tool (XDT) is now included in the *RSX-11M-PLUS and Micro/RSX XDT Reference Manual.*
- Terminal Emulation and File Transfer With Other Systems is now included in the *RSX-11M-PLUS and Micro/RSX System Management Guide.*

Creating an Optional Software Package

The following sections have changes:

- Creating a Diskette Package

  The section on creating a diskette package has been revised to reflect a new INSTALL.DAT statement, as follows:

  ```
  SYSTEMVERSION=vrsn
  ```

- Creating a Tape Package

  The sections concerned with creating a tape package have been revised to reflect changes made in the guidelines.

  The sections describing how to create an optional software tape kit refer to using two input devices for a group of files to be copied to a backup set on the tape. The Backup and Restore Utility (BRU) will only accept two or more input devices if they contain a multivolume backup set on disk or tape.

Chapter 1
The Program Development Environment

This chapter introduces the software and hardware that you need to develop programs using the PDP-11 MACRO-11 assembler on a Micro/RSX multiuser system. The remaining chapters in the guide further describe and illustrate how to use the tools and facilities introduced in the following sections.

1.1 Software Tools

The Micro/RSX system makes software tools available as executable programs called system tasks. The system tasks include the DIGITAL Standard Editor (EDT), the PDP-11 MACRO-11 Assembler (MAC), the Task Builder (TKB), several aids to debugging, and a number of utility tasks. These tasks combine to form the program development environment. Some of these tasks, like EDT, are already part of your software environment as a result of installing the Micro/RSX Base Kit. The other software necessary for doing MACRO-11 programming, like the PDP-11 MACRO-11 assembler, is included in your Micro/RSX Advanced Programmer's Kit. The Advanced Programmer's Kit includes the following software options:

- Advanced MACRO-11 Program Development

PDP-11 MACRO-11 assembler	MACRES.TSK
File Dump Utility	DMPRES.TSK
File Compare Utility	CMPRES.TSK
Resource Monitoring Display	RMD.TSK
File Structure Verification Utility	VFYRES.TSK
Postmortem Dump Utility	PMDRES.TSK
System Macro Library	RSXMAC.SML

RMS-11 macro library	RMSMAC.MLB
File control system subroutines	NOANSLIB.OLB
Virtual memory subroutines	VMLIB.OLB
On-Line Debugging Tool (ODT)	ODT.OBJ and ODTID.OBJ
Sample indirect command files	INDSYS.CLB

- Privileged Program Development

Loadable Executive Debugging Tool (XDT)	XDT.TSK
Loadable RL02 crash driver	DLCRSH.TSK
Loadable DU-type device crash driver	DUCRSH.TSK
Crash Dump Analyzer (CDA)	CDARES.TSK
Error log compiler	CFLRES.TSK
Error logging auxiliary	ERRLOGETC.ULB
Control File Language source file	DEVSM1.CNF
Executive macro library	EXEMC.MLB
Executive object library	EXELIB.OLB
Executive prefix assembly file	RSXMC.MAC
Executive map file	RSX11M.MAP
Magnetic tape template INS file procedure	TAPEINS.CMD

- File Transfer and Terminal Emulation Support

Micro/RSX File Transfer Utility	MFT.TSK
Data Terminal Emulator Utility	DTE.TSK

- Magnetic Tape Software Support

Loadable MS-type device crash driver	MSCRSH.TSK
Loadable MU-type device crash driver	MUCRSH.TSK
RMS-11 File Backup Utility	RMSBCK.TSK
RMS-11 Restoration Utility	RMSRST.TSK
Magnetic tape Ancillary Control Processor (ACP)	MTAACP.TSK
Magnetic tape ACP message facility	F11MSG.TSK
Magnetic tape control processing task	MAG.TSK
File Transfer Utility Program	FLXRES.TSK

1.1.1 DIGITAL Command Language

The command language used on Micro/RSX systems is the DIGITAL Command Language (DCL). This command language is used to communicate with the system and to invoke system services.

You are not required to use the full form of DCL commands. Usually, you need type only the command elements required to form a unique command. For clarity, most examples in this manual appear in full format. You can always shorten any command or qualifier to four characters. Most commands and qualifiers can be entered with even fewer characters.

DCL prompts you for all required command elements. If you do not understand a prompt, type a question mark (?). DCL will print HELP text explaining the format and function of the command, and then prompt you for required input.

See the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2* for a complete description of how to use DCL.

1.1.2 EDT Text Editor

The Micro/RSX system includes the text editor, EDT, which you use to create a source code file. This editor is an interactive editor that enables you to enter American Standard Code for Information Interchange (ASCII) text at a terminal and store the text in a disk file. It also lets you access text in a disk file; examine, delete, and change text; and insert new text. The disk file is then used as input to other tasks involved in the program development process.

See the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2* for a complete description of EDT.

1.1.3 PDP-11 MACRO-11

The programming language distributed with the Advanced Programmer's Kit is the PDP-11 assembly language, MACRO-11.

MAC is the task that assembles MACRO-11 language files. It accepts a disk source input file in ASCII format and can create a relocatable object module and a listing file of the source language. The object module contains all the object records and relocation information needed to link with other object modules. All symbol definition done by the assembler has a base address of 0. The allocation of virtual addresses and relocation is left for the task-building process.

You invoke MAC with the MACRO command. Chapter 3 provides a basic explanation of how to assemble a program module using the MACRO command. Appendix A provides a detailed description of all the optional command qualifiers for the MACRO commmand. The following subsections provide an overview of PDP-11 MACRO-11.

1.1.3.1 Source Input

Source input to MACRO-11 consists of free-format statements; each line of input contains a single statement. Input statements are either PDP-11 instructions, MACRO-11 assembler directives, macro calls, or direct assignments. Statements can contain labels that allow control to change locally (within the module) or that enable control to be passed between modules (globally).

Source input usually contains user-defined symbols, which are either local or global. A local symbol is defined in the current source file and is referenced only within the current file. A global symbol is defined in one source file but can be referenced in one or more other source files.

The assembler allows you to use both local and global symbols as labels for statements. When a global symbol appears as a label, the related statement is referred to as an entry point (that is, a point at which other modules can transfer control to the current object module). You can use local symbols as statement labels to define points to which control transfers within an object module.

The assembler evaluates all local symbol definitions in a source file. Any symbols remaining undefined are classified as global. Thus, after an assembly, all local symbols are assigned relative locations, but the module may contain references for which definitions must be supplied from outside that module. The resolution of these references is left for the task-building process.

1.1.3.2 Source File Directives

Assembler directives in a source file allow you to perform operations such as the following:

Program sectioning
Allows code or data within an object module to be overlaid by, or concatenated with, code or data in other object modules or in noncontiguous locations within the same module. Program sectioning is especially useful when convenient physical ordering differs from logical reference ordering (for example, in table-generating macro statements).

Listing control
Enables documentation features that generate headings, page formats, and a table of contents.

Conditional assembly
Allows optional omission or inclusion of lines of code or user-defined symbols.

Data storage
Controls the size and content of data areas.

Special statements called macro directives allow you to reference a predefined symbol that causes the assembler to expand a single line source statement into multiple lines of code or data and insert the assembled result in the object module. Such macro symbols are typically used for recurring coding sequences. The insertion of the code sequence occurs at each point where you refer to the macro symbol. Definitions for such macro symbols can occur in the source file itself or can reside in a macro library. Generally, you place infrequently used macro definitions in the source file that invokes them, and you store frequently used macro definitions in a macro

library. The Executive and file-processing services are made available to the program through macro symbols that are defined in a DIGITAL-supplied macro library.

1.1.3.3 How the Assembler Works

MACRO-11 is a two-pass assembler. During the first pass, the assembler classifies all the symbols into local and global groups, performs statement generation, locates all macro symbols, and, if necessary, reads the macro definitions from libraries. At the end of the first pass, the assembler must have processed all local references (such as all undefined global symbols) that are to be resolved by TKB.

During the second pass, the assembler actually generates the object module and listing files, flagging with an error code in the listing file those source statements containing errors. If you requested a cross-reference listing of symbols, the assembler also generates a request for the Cross-Reference Processor (CRF) to create the proper information. (CRF is explained in Section 1.1.6.1.)

The MACRO-11 listing file provides documentation for the module and serves as a tool for debugging the code. As a reference aid, the assembler generates and includes line numbers in the listing for each statement in the source file. In addition, the listing includes a symbol table showing symbols, their attributes, and their values (if known at assembly time).

As a debugging aid, the listing also provides a current location counter for each program section defined in the source file. The location counter value is important in debugging because it provides the offsets into the module for each program section. An offset, combined with the base load address for a program section (from the TKB map), allows you to access locations in the memory-resident task image during debugging.

1.1.4 Task Creation

The Micro/RSX Task Builder (TKB) is a multipurpose tool that you invoke using the DCL command LINK. TKB allows you to create a loadable program (called a task image), define and structure a shared area of memory (called a resident common), and arrange shareable routines to reside in memory (called resident libraries). Although TKB has many complex aspects, this guide introduces only TKB's most common use: building a task image. (See the *RSX-11M-PLUS and Micro/RSX Task Builder Manual* for more information on TKB.)

To build a task image, TKB accepts, as basic input, the output of a language processor: an object module or modules. TKB can optionally generate a file of executable code (the task image), a file of memory allocation information (a map), and a special file of symbol definitions used in constructing the task (the symbol definition file). The task image, residing on disk, is in a format suitable to be loaded into memory and executed. If you generate a cross-reference listing, the listing itself contains only global symbols and is appended to the map file.

In creating a task image, TKB's primary functions are linking, address binding, and building system data structures. Linking involves resolving global references in all object modules and resolving program section references among all object modules. Address binding is assigning virtual address space within the task. Building system data structures involves creating elements that the system requires to load the task image into memory and to execute the task. To resolve global symbols that are not defined in any of the input object modules, TKB searches any object libraries you specify and, as a default condition, searches the system object library.

Because a PDP-11 processor can address only 32K words (the address limit of 16 bits) at any one time, a task cannot reference more than 32K words at a time. However, if you use certain advanced programming techniques, TKB allows a task to access more code or data than can fit within the address limits. Techniques to overcome the addressing limits include the following:

- Overlaying segments of a task with either disk-resident or memory-resident code
- Mapping to different regions of memory outside the physical limits of the current task space

For more information on these techniques, refer to the *RSX-11M-PLUS and Micro/RSX Task Builder Manual.*

The memory allocation information, or map, produced by TKB shows you how program sections are arranged in task memory (their starting virtual addresses and extents on mapped systems and physical addresses and extents on unmapped systems), what contributions are in a program section, any undefined symbols, and the optional cross-reference listing of global symbols. You can use the starting virtual addresses, combined with the current location counter values (provided by the assembler), as offsets to access locations within the memory-resident task during debugging.

1.1.5 Debugging Aids

This section introduces the debugging aids described in this guide and provided with Micro/RSX operating systems to assist in identifying faulty code.

1.1.5.1 On-Line Debugging Tool

The On-Line Debugging Tool (ODT) allows interactive control of task execution. You specify to TKB that you want a debugging aid included in a task. TKB inserts the module LB:[1,1]ODT.OBJ into the task.

When you run a task that includes ODT, execution begins at the ODT transfer address rather than at the task starting address. Therefore, ODT gains control and allows you to type special commands that establish base addresses and that set breakpoint locations within the task. After you tell ODT to begin task execution, ODT saves the instructions at the breakpoint locations you specified and replaces them with PDP-11 BPT instructions. Upon encountering a BPT instruction in the task, the Executive passes control to ODT at its breakpoint routine. ODT saves task registers in special locations, restores instructions to the breakpoint locations, and transfers control to the user task terminal. By typing ODT commands, you can examine and alter any instructions or data within task memory.

ODT also enables the BPT synchronous system trap (SST) entry point in the task. If a task generates an SST error, ODT gains control at its SST entry point, prints a notice at the user terminal, and passes control to the terminal. You can use the ODT commands to discover the cause of the error, correct it, and perhaps continue executing the task.

To successfully modify instructions, you must have a thorough understanding of the PDP-11 instruction set. If you are programming in a high-level language, you should avoid interactive debugging whenever possible. Refer to Chapter 5 for more information about ODT.

1.1.5.2 Postmortem Dump

The Postmortem Dump (PMD) is an installed task that is directed by the Executive to extract run-time-related data about a terminated task, format it, and request a printed listing. Normally, when a task generates an SST, such as that caused by an improper reference to an odd address or a reference to a nonexistent memory location, the Executive tries to transfer control to an SST entry point defined by the task. If the task does not have an SST routine defined for the particular type of trap, the Executive aborts the task.

To terminate the task, the Executive performs an abort operation and notifies the Task Termination Notification program (TKTN), which displays, on the user terminal, the reason for the termination and the contents of the task registers. Without PMD, you can acquire no further information about the task. Commands do exist that allow you to fix a task in memory and physically examine the contents of the task image. However, this is a complicated procedure and beyond the scope of this book.

By enabling a PMD for a task that itself does not handle SSTs, you tell the Executive to supply more data at abnormal task termination. That is, the Executive follows the abort procedure and, in addition, creates a request for PMD to create the dump. PMD examines system and task structures to preserve status and run-time data, reads the task image from memory, and writes it to disk in a readable format. PMD then queues a request to print the file containing the dump data, after which the Executive completes the task abort procedure.

1.1.5.3 Snapshot Dump

The snapshot dump ($SNAP), also using PMD, generates an edited dump of a running task. Because the $SNAP requires you to insert special code (for example, the $SNAP macro call) in a task, it is more difficult to use than PMD. However, by inserting the snapshot dump code in the task, you can choose the location at which the dump is created and select the extent and format of the dump. In addition, you can generate the dump from more than one location and, therefore, as many times as needed during task execution.

It is often useful to include debugging facilities such as $SNAP in your task based on defining a conditional variable. To include the facility while you are debugging, simply define the variable. You can then omit the facility by reassembling the code with the conditional variable undefined.

Refer to the following manuals for more information on debugging aids:

- *RSX-11M-PLUS and Micro/RSX Debugging Reference Manual*
- *RSX-11M-PLUS and Micro/RSX Crash Dump Analyzer Reference Manual*

1.1.6 General Utilities

This section introduces the general-purpose utility programs that are discussed in this guide.

1.1.6.1 Cross-Reference Processor

The Cross-Reference Processor (CRF) is an installed task that receives requests from the MACRO-11 assembler and TKB to generate cross-reference listings of symbols. CRF generates a specially formatted file containing the cross-reference data and appends that file to the assembler listing or the task map file. Therefore, if you request a cross-reference listing of symbols, it always appears at the end of a listing or map file.

A request for the services of the CRF is included in your command line to the PDP-11 MACRO-11 assembler and TKB; use the /CROSS_REFERENCE qualifier to the MACRO and LINK commands. (For a technical description of CRF see the *RSX-11M-PLUS Utilities Manual.*)

1.1.6.2 Library Operations

The LIBRARY command creates and maintains specially formatted library files on disk: one for macro call definitions and one for object module subroutines. The LIBRARY command can also create a universal library file to contain any file type you prefer.

The PDP-11 MACRO-11 assembler and TKB can access the macro and object library files, respectively, and extract the proper code from them. Libraries are convenient to use because they encourage sharing of code, provide faster access to multiple modules (only one file need be opened and closed), occupy less space than the equivalent number of separate modules, and impose a coding standard. The library files you create using the LIBRARY command are in the same format as those that DIGITAL supplies with the Micro/RSX operating system.

1.2 System Software

DIGITAL supplies system software in two standard library formats: macro call definitions and object module subroutines. You use macro libraries as input to the assembler and object libraries as input to TKB. The following two sections describe these system libraries.

1.2.1 System Directives—Macro Libraries

DIGITAL makes available system directives and system-related features through calls; definitions for these calls reside in macro libraries. The libraries are stored in a predefined file area known as the system directory. The system directory is [1,1] on the system library device (referenced explicitly by the device-independent designation LB). Table 1–1 summarizes the macro libraries that DIGITAL supplies.

Table 1–1: DIGITAL-Supplied Macro Libraries

File Name and Type	Description of Contents
RSXMAC.SML	System Macro Library. Contains the macro definitions for all system directives and File Control Services (FCS) file-processing calls. Default library for the assembler.
EXEMC.MLB	Executive Macro Library. Contains the symbol and offset definitions for the Executive data structures.
RMSMAC.MLB	PDP-11 Record Management Services (RMS-11) Library. Contains the definitions for RMS-11 calls for sequential, indexed, and relative file I/O.

To use these libraries, you should follow the specific procedures described in the system documentation. Typically, you supply in the source code the appropriate names of the modules as parameters of a .MCALL MACRO-11 directive. This action tells the assembler to generate an entry for that call in its macro symbol table and to search the appropriate library for the definition of the macro symbol.

In translating source code, the assembler first checks for macro symbols. When the assembler finds an operator on a source line, it searches its macro symbol table to see whether the operator is a macro symbol. An operator is any PDP-11 operation code, MACRO-11 assembler directive, or macro symbol. If the operator is a macro symbol, the assembler applies the local definition for the macro symbol or extracts the definition from a library you specified or from the system library. By searching the user-supplied library first, the assembler allows you to tailor the definitions of system macro calls or PDP-11 instructions. MACRO-11 assembles the macro definition with any accompanying parameters and includes the assembled code in the object module. As a result, the proper code is included from a library.

The System Macro Library, RSXMAC.SML, provides you with the code that enables a task to issue system directives and to obtain File Control Services (FCS). These services enable a task to obtain run-time and system information, perform I/O functions, communicate with other tasks, manipulate logical and virtual address space, control execution, and properly exit. In general, most system features are made available to a task through macro calls to the System Macro Library. For the System Macro Library RSXMAC.SML, you need not designate the library name to the assembler. As a default condition, the assembler automatically searches the System Macro Library.

The Executive macro library, EXEMC.MLB, provides you with code to allow software to refer to offsets within the Executive and system definitions of the Executive data structures. This library is provided for assembling privileged tasks and for incorporating specially written device drivers onto the system. This topic is covered fully in the *RSX-11M-PLUS and Micro/RSX Guide to Writing an I/O Driver.*

The Record Management Services library, RMSMAC.MLB, is provided to support file and record access to RMS-11 data. RMS-11 is an upward-compatible extension of FCS and offers more functions, such as indexed sequential (keyed) access to data. You include the RMS-11 macro symbols in the source code and supply to the assembler the name of the RMS-11 library to use. The assembler extracts the definitions from the library and includes the RMS-11 code in the object module.

1.2.2 System Subroutines—Object Libraries

System object libraries provide general utility functions and special-purpose Executive features. These libraries, like the macro libraries, reside in system directory [1,1] on the system library device (LB). Table 1–2 lists and describes the object libraries that DIGITAL supplies.

Table 1–2: DIGITAL-Supplied Object Libraries

File Name and Type	Description of Contents
SYSLIB.OLB	System Library. Contains register handling, arithmetic functions, data conversion, and output formatting. Default library for TKB.
VMLIB.OLB	Virtual Memory Management Library. Contains dynamic memory, core allocation, virtual memory, and page management subroutines.
EXELIB.OLB	Executive Library. Contains the definitions of the Executive symbols.
NOANSLIB.OLB	Another version of SYSLIB. Contains FCS, which include big buffering support and command line processing subroutines.
RMSLIB.OLB	Record Management Services Library. Contains the routines for sequential, relative, and indexed I/O.

You include system object subroutines in a task by specifying the subroutine name as the operand of a CALL macro or Jump to Subroutine (JSR) instruction in the source code. The language processor, at the point of the reference, generates the instructions to transfer control to the external subroutine. The name of the subroutine is left as an externally defined global symbol for TKB to resolve.

To ensure that subroutines are placed in the task image, TKB (as a default operation) searches the library SYSLIB.OLB for subroutine names that remain undefined after the search of any user-specified libraries. TKB attempts to match the undefined global reference (the subroutine name in a module) with an entry point name in the SYSLIB.OLB library. When it finds a match, TKB extracts a copy of the module defining the symbol from SYSLIB.OLB and inserts the subroutine in the task image. Any further references to that symbol in the task are defined by the subroutine, and TKB need not add any code to resolve further references.

If a module references subroutines that are in an object library other than SYSLIB.OLB, you must specify that library when you build the task. TKB performs the same search operations on user-supplied libraries as it does on the default search of SYSLIB.OLB. TKB also searches any user-specified libraries, in the order in which you specify them, before it searches the system library. If you intend to build FCS into the task, you must use the NOANSLIB.OLB library intead of SYSLIB.OLB.

1.3 Hardware

You need the following three types of devices for program development:

- Disks
- Terminals
- Printers

This section briefly introduces these devices and tells where you can find further information. In general, each hardware unit on the system is delivered with relevant hardware documentation that provides programming information in addition to operational instructions. Your installation should have a library of such hardware documentation. If you are not writing any specially

tailored software for these devices, the system software handles them transparently through such mechanisms as the print spooler and DCL commands.

1.3.1 Disks

Disks are the main storage media on Micro/RSX systems. The minimum hardware configuration for a MicroPDP-11 running the Micro/RSX operating system includes a disk storage subsystem that combines a fixed disk drive with a dual diskette drive. The fixed drive is an RD51-A fixed disk with 10 megabytes (Mb) of storage, and the diskette drive is an RX50-AA that provides two access slots for single-sided 400-kilobyte (Kb) 5.25 flexible diskettes. The RD51 serves as the system disk, and the RX50s are used for data devices and backup.

This disk configuration is a basic application tool and not a recommended storage solution for doing serious program development work, such as designing and developing application programs. To use Micro/RSX for program development you should increase disk storage capacity by adding on another RD51; an RL02, which gives an additional 10 Mb to 40 Mb of storage that is also removable (you can have up to four RL02 drives on a controller); an RD52; or an RD53 drive.

1.3.2 Terminals

Terminals are the means by which you communicate with the system. Some DIGITAL terminals handle 7-bit ASCII characters (system software usually ignores any eighth, or parity, bit). The VT200-Series terminals handle 8-bit ASCII characters. You perform input to the system through a typewriter-like keyboard; the system returns output to you either on a screen at a video-display terminal or on paper at a hardcopy terminal. Video-display terminals are more convenient because they typically operate at faster rates than hardcopy devices. Hardcopy terminals, however, have the advantage of providing a record of what transpired during a session on the system.

Terminals are connected to the computer through either a direct line or a modem unit over a dial-up telephone line. If you are not familiar with using a terminal, you should read the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2*. These manuals explain how to access the system and basic system functions using DCL.

1.3.3 Printers

Printers provide hardcopy output of data. In Micro/RSX, you communicate with the printer through the Queue Manager (QMG) subsystem by using the DCL command PRINT (see the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2* for more information on the QMG). All systems have a terminal or other output device serving as a line printer. All listings from the PDP-11 MACRO-11 Assembler or TKB are queued to the system line printer.

1.4 Overview of the Program Development Process

Figure 1–1 illustrates the steps in the program development process. The figure and the following paragraphs briefly describe these steps, which are treated in greater detail in Chapters 2 to 7.

The steps normally taken to prepare a program to run on the system are as follows:

1. Create a source program in a file on disk.
2. Submit the source file to the PDP-11 MACRO-11 assembler to produce an object module.
3. Submit the file (or files) containing the object modules to TKB to create a file containing a loadable task image.
4. Request the Executive to execute the task.

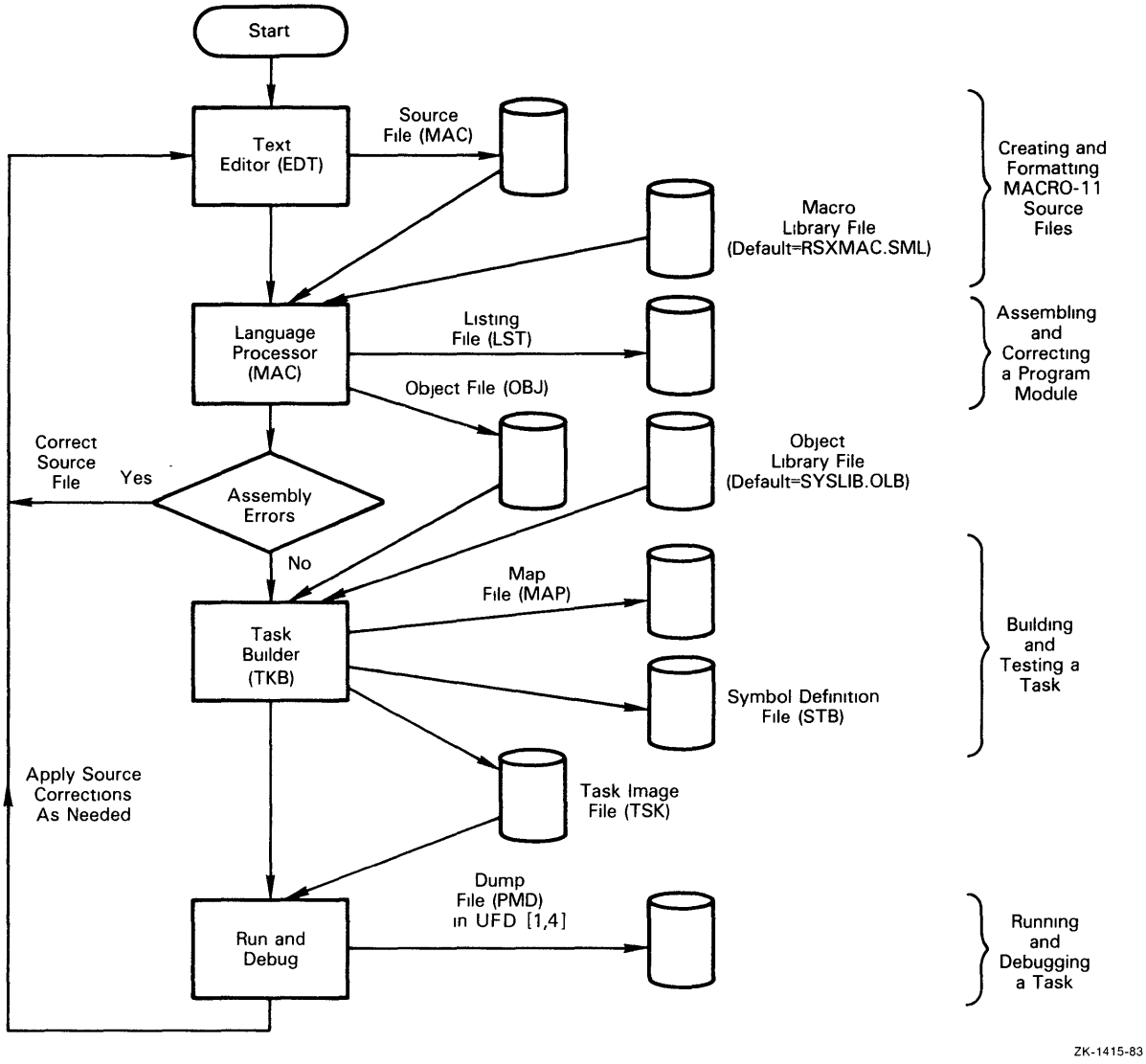
You use a text editor to create the source file. This guide suggests a skeleton format for source files and shows how to replicate and modify the skeleton file. The skeleton file becomes a common base from which you create each new source file.

The assembler creates the file of relocatable object code and also accesses the System Macro Library to include code for system directives in the object file.

TKB creates the file of loadable code, assuming certain default conditions about the run-time environment and building these characteristics into the task. TKB also accesses system and user-specified libraries to resolve references in the task.

Once you have a task image, you request the Executive to run the program. If any errors are encountered, you must edit the source file, reassemble or recompile, build a new task image file, and try again.

Figure 1–1: The Program Development Process



Chapter 2
Creating MACRO-11 Source Files

Your first step in program development is to create a file that contains MACRO-11 source statements. One way to do this is to create a skeleton source file that you can use as a framework for all your source programs. This chapter describes a source file format that you can use as a guideline to create your own skeleton file, presents MACRO-11 statements to include in the file, and explains elementary editing commands that you can use to create and modify source files.

DIGITAL has established a coding standard to enhance the readability and maintainability of its MACRO-11 source programs. That standard is outlined in the *PDP-11 MACRO-11 Language Reference Manual.*

2.1 MACRO-11 Skeleton Source File Format

This section presents the skeleton and source statement formats and discusses each of the elements in the skeleton. Figure 2–1 illustrates the basic elements of the skeleton: a preface, definitions, functional descriptions, and the code itself.

The source file preface, or preamble, should be on the first page. The preface does the following:

- Describes the code
- States its ownership
- Identifies the author
- Defines the changes to the code
- Provides a brief description of the module's function

The details of the code should follow the preface of the module. Declarations, such as local symbol, macro, and data definitions, should appear toward the front of the code to make reading the code easier. Before each routine in the module, you should place detailed information that does the following:

- Describes the function of the routine
- Defines the required input to the routine

- Describes the product of the routine
- Describes the effect of the routine upon execution

Each statement line in a source file should follow a consistent format. To create readable, consistent code that is easy to trace, it is recommended that you format your statements according to the following definitions:

Label
A user-defined symbol that identifies a reference location in the code.

Operator
A PDP-11 operation code, MACRO-11 assembler directive, or macro symbol.

Operand
An argument (or arguments) or parameter (or parameters) of an operator.

Comments
Information you provide to describe what effect you desire from the execution of the instruction. Comments do not affect program execution; the assembler merely transfers them to the listing file produced during the assembly.

Figure 2–1 illustrates the format of a MACRO-11 source file. Figure 2–2 illustrates the format of a MACRO-11 source statement.

Figure 2–1: MACRO-11 Source File Format

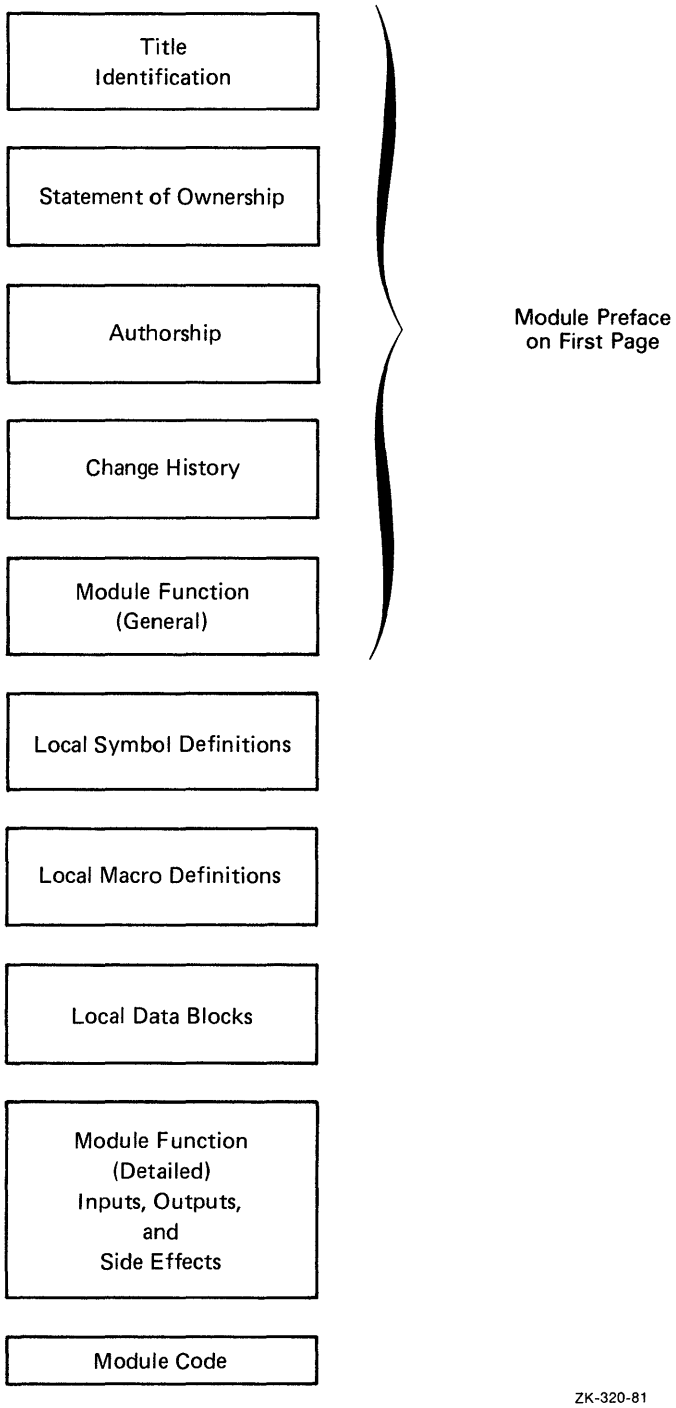
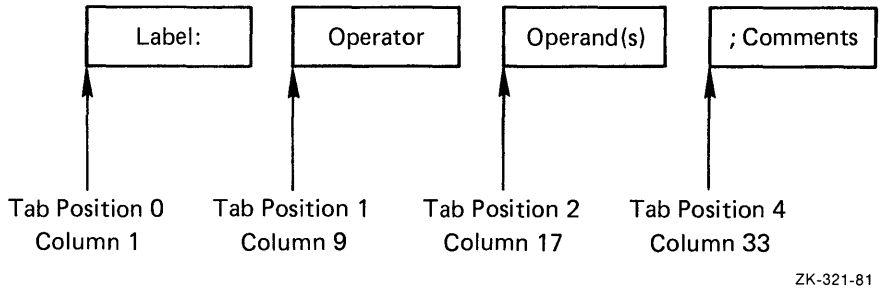


Figure 2–2: MACRO-11 Source Statement Format



Comments, accompanied by selected MACRO-11 assembler directives, constitute the source file skeleton. This skeleton provides the structure on which you build the source file. Directives in the source file skeleton identify the code and control the format of the listing. Example 2–1 shows a sample skeleton. Detailed descriptions of the parts of the source file skeleton follow the example.

Example 2-1: Sample Skeleton Source File

```
        .TITLE  SKELTN SOURCE FILE SKELETON  ❶
        .IDENT  /01/  ❷
;
;
; AUTHOR: Z  ❸
;
;
; CHANGES:  ❹
;
;
; MODULE FUNCTION GENERAL:  ❺
;
;
        .PAGE                   ; BREAK PAGE FOR PREFACE  ❻
        .SBTTL  SYMBOL, MACRO, DATA DEFINITIONS  ❼
        .LIST   TTM  ❽
        .NLIST  BEX             ;SUPPRESS BIN EXTENSION  ❾
        .MCALL  EXIT$S          ;EXEC'S EXIT MACRO  ❿
;
; LOCAL SYMBOL DEFINITIONS:  ⓫
;
;
; LOCAL MACROS:  ⓬
;
;
; LOCAL DATA BLOCKS:  ⓭
;
        .PSECT  DATA,D,RW  ⓮
;
; MODULE FUNCTION DETAILED:  ⓯
;
;
;       INPUTS:
;
;       OUTPUTS:
;
;       SIDE EFFECTS:
;
; START CODE HERE
        .PAGE
        .SBTTL
        .PSECT
START:
END:    EXIT$S                  ;EXIT CLEANLY TO EXEC
        .END                    ;TELL ASSEMBLER END OF CODE  ⓰
```

Notes on Example 2–1:

❶ .TITLE Directive

The .TITLE directive allows you to name the module. The assembler takes the first six nonblank (alphanumeric) characters, up to the first blank or horizontal tab character, as the module name. Following the name in the .TITLE directive, you can use up to 24 characters to describe the function of the module. The name and the description appear as the first entry in the header line of each page in the assembly listing. For example, consider the following .TITLE directive:

```
.TITLE SKELTN SOURCE FILE SKELETON
```

The assembler takes the characters SKELTN as the module name. The remaining characters up to the 30th character are taken as the description. Any remaining characters after the 30th character would be discarded.

The assembler does not relate the name you specify in the .TITLE directive to the name you specify for the source or object files. To minimize confusion, however, it is helpful to apply the name specified in the .TITLE directive to the source file. (Note that the sample code and commands shown in this guide use different names to help you distinguish their usage.)

The name the assembler extracts from the .TITLE directive is also important in subsequent steps of program development. The Task Builder (TKB) lists this name in its memory allocation synopsis to show which object modules made contributions to each program section in the task image. In addition, if the LIBRARY command is used to insert the object module in an object library, this name is kept in the directory of the library to refer to the object module.

❷ .IDENT Directive

The .IDENT directive records the version of the module. You can establish your own version identification conventions. The identification follows the module into the task image and is displayed in the map. Knowing whether the correct version of the module was linked into the task image helps in the debugging and maintenance process.

❸ Author Line

The author line identifies the originator of the code.

❹ Changes

This section of the source file describes any modifications that have been made to the module. You can develop a convention whereby the author's initials and a number can indicate a change. The author of the modification can identify the change in this section and flag each line of code with an additional comment, such as the following:

```
; TOM JONES      8-AUG-86        1.01
; TJ001          ADD STATE TAX TO TOTAL
```

A changed or added line in the code can be flagged with the notation TJ001 as follows:

```
ADD   A,B            ;TOTAL WITH TAX   ;TJ001
```

This procedure helps the author recall what changes were made to the module and assists others in determining the extent of changes.

❺ Module Function General

In the module function part of the source file, you can describe the general processing operations that the code performs. This description can include how the module relates to the system or specific application, that is, what type of processing precedes and follows the execution of this module.

❻ .PAGE Directive

The .PAGE general-purpose directive causes a page break in the assembly listing. It appears as shown to keep the preamble alone on the first page of the listing (after the table of contents). You can use the .PAGE directive throughout the module to generate page breaks for different subroutines.

❼ .SBTTL Directive

The .SBTTL general-purpose directive creates an entry for the assembly listing table of contents printed at the front of the listing. A table of contents is helpful in summarizing the subroutines in a large module. Therefore, the text you supply with the directive should describe what the related subroutine does. In addition to appearing in the table of contents, the text appears on the second line of the heading at the top of each listing page. If your modules typically contain only a small number of subroutines, you probably will not find the table of contents feature very useful.

❽ .LIST TTM Directive

The .LIST TTM general-purpose directive creates a listing formatted more conveniently for output on a terminal. (Chapter 3 of this guide shows how to display a listing at a terminal.) You can include the directive during the early stages of program development and later remove it from the stabilized code.

❾ .NLIST BEX Directive

The .NLIST BEX general-purpose directive suppresses the binary extension of statements beyond what can fit on one source statement line. Using this directive saves excess printing in the assembly listing. For example, only the binary value of the first characters of an ASCII string would appear in the listing. The directive simply makes the listing more readable and saves paper.

❿ .MCALL Directive

The .MCALL general-purpose directive is the means by which you tell the assembler the names of the externally defined macro calls that appear in the source file. The directive causes the assembler to create entries in its macro symbol table for the macro names and to look up the definitions of the related calls in either a user or a System Macro Library. The assembler includes the definitions from the library in the module where the calls themselves appear.[1] The EXIT$S directive (shown in the .MCALL statement) should be in every user program for a clean exit. It is the last statement the program (task) executes before it returns control to the Executive. (The EXIT$S directive performs important system housekeeping operations for the task.) The related definition for EXIT$S resides in the file RSXMAC.SML in system directory [1,1] on the library device (LB). DIGITAL recommends that user tasks

[1] If you do not include the directive .LIST ME (list macro expansions) or .LIST MEB (list macro expansion lines that generate object code) in the source file, the assembler does not insert in the listing the expanded source code of the macros it assembles.

exit by using the EXIT$S directive. (An alternative form of exiting allows a task to exit and post status.)

If a call for an externally defined macro statement appears in the source file but is not preceded by a .MCALL directive and the macro name, the assembler treats the unrecognized macro call as an implicit .WORD data storage directive. (If the macro call has parameters, the assembler may generate an error because of illegal syntax for a .WORD directive.) Later, when you build the task with the related object module and the macro name is not a valid symbol, TKB flags the name as an undefined reference. Thus, without the .MCALL directive, the assembler does not know that it must search libraries to resolve the macro symbol.

⓫ Local Symbol Definitions

In this section of your source file, you collect symbols in direct assignment statements. Because symbols in MACRO-11 can be defined as expressions of other symbols, having the definitions in one place is an advantage. In addition, good programming practice encourages using symbols instead of simply supplying a numeric constant.

For example, in defining a 10-byte buffer, the best method is to define a symbol and then use the symbol in the buffer definition, as follows:

```
;
; LOCAL DEFINITIONS
;
SIZB = 10.
.
.
.
;
; LOCAL DATA BLOCKS
;
BUFB:    .BLKB    SIZB
```

This method has the following advantages:

- If a single constant that is referred to in numerous places in the code must be altered, you need perform only one edit (to the symbol definition) to effect the change.
- If all the symbols are gathered in one place in alphabetical order, reading the code is simplified.
- You can find all references to a symbol in a cross-reference listing. The cross-reference capability allows you to examine all the references to a symbol and confidently assess the effects of altering the symbol definition.

These advantages are lost if you use constants. Thus, the symbol list would contain such local symbol definitions as SIZB = 10. The symbols themselves would appear in the module code.

⓬ Local Macro Definitions

The definition of a macro statement can appear anywhere in the source file as long as the definition appears before the first occurrence of the macro statement. It is better programming practice to place all macro definitions in a standard place near the front of the source file.

⓭ Local Data Blocks

This section of the source file defines such data as buffers, status words, and status bytes. Generally, it describes the local storage that the module references. It is good programming practice to use a separate .PSECT directive for data.

⓮ .PSECT Directive

The .PSECT directive establishes a name and attributes for a program section. A program section is a unit allocation of memory reserved for either code or data. For example, you can establish a program section to contain data for your program as follows:

```
.PSECT DATA,D,RW
```

The .PSECT directive creates the program section named DATA with the attributes data (D) and read/write (RW). You may give a program section for data either the read-only (RO) or the read/write (RW) attribute. (The assembler applies other default attributes not relevant to this discussion.) Consult the *RSX-11M-PLUS and Micro/RSX Task Builder Manual* for a discussion of program section allocation in multiuser tasks.

The three most important aspects of the .PSECT directive are as follows:

- Contributions defined for a specific program section can be in separate places in a source file or in separate source files.
- Attributes of the program section are passed to TKB.
- Contributions for a specific program section with the same attributes are collected in one continuous allocation of memory space by TKB.

In the skeleton file, it is useful to define one program section to contain the data elements referenced in the task and to define another program section to contain the code.

⓯ Module Function Detailed

This section of the source file can be as general or specific as necessary to describe the functions of the module. A complex module should have a lengthy discussion; a simple module need not have as much. At the minimum, this section should state the register usage on input to and output from the module.

⓰ .END Directive

The .END directive in a module signals the logical end of source input and optionally specifies the task transfer address. The transfer address is the location at which program execution begins. Although each source file should contain a .END directive, only one source file should define the transfer address. The assembler does not process lines beyond the one on which the .END directive appears.

2.2 Creating a Source File from a Skeleton File

This section describes how to use the EDT text editor to create a skeleton file and then to create a source file from the skeleton. For more detailed information on using EDT, see the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2*.

2.2.1 Performing the Initial Input

To create the skeleton file, type EDT and the specification of a new file, that is, one that is not in your directory.

Example

```
$ EDIT [RET]
File? SKEL.MAC [RET]
Input file does not exist
[EOB]
*c [RET]
```

Runs the editor, determines that the file does not exist, creates the file, and enables keypad editing.

Type the input according to Example 2–2. Leave any typographical errors until after you have become familiar with the editing commands described in the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2*. The notation conventions appearing in Example 2–2 are described in the Preface at the front of this guide.

2.2.2 Inserting Blank Lines in Text

To insert a blank line in the source file, as shown in Example 2–2, press the space bar or the TAB key on a new line followed by the RETURN key. Thus, to enter a blank line, you need press only one nonprinting character, such as the TAB key, on a new line.

2.2.3 Exiting from EDT

To exit from EDT, first press the GOLD key and then the COMMAND function key. Type EXIT after the prompt and press the ENTER function key. EDT saves the file that you have typed.

When EDT exits, it prints the file specification and returns control to the operating system. The dollar sign prompt ($) indicates that DCL is ready to accept a new command.

Example 2–2: Creating the Skeleton File SKEL.MAC

```
$ EDT SKEL.MAC [RET]
Input file does not exist
[EOB]
*c [RET]
[TAB]      .TITLE [TAB] SKELTN SOURCE FILE SKELETON [RET]
[TAB]      .IDENT [TAB] /01/ [RET]
; [RET]
; [RET]
; AUTHOR: Z [RET]
; [RET]
[TAB][RET]
; [RET]
; CHANGES: [RET]
; [RET]
[TAB][RET]
; [RET]
; MODULE FUNCTION GENERAL: [RET]
; [RET]
[TAB][RET]
; [RET]
[TAB]      .PAGE [TAB] [TAB] [TAB]          ; BREAK PAGE FOR PREFACE [RET]
[TAB]      .SBTTL [TAB] [TAB] [TAB]         ; SYMBOL, MACRO, DATA DEFINITIONS [RET]
[TAB]      .LIST [TAB] TTM [TAB] [TAB]      ;TERMINAL LISTING MODE [RET]
[TAB]      .NLIST [TAB] BEX [TAB][TAB]      ;SUPPRESS BIN EXTENSION [RET]
[TAB]      .MCALL [TAB] EXIT$S [TAB][TAB]   ;EXEC'S EXIT MACRO [RET]
[TAB][RET]
; [RET]
; LOCAL SYMBOL DEFINITIONS: [RET]
; [RET]
[TAB][RET]
; [RET]
; LOCAL MACROS: [RET]
; [RET]
[TAB][RET]
; [RET]
; LOCAL DATA BLOCKS: [RET]
; [RET]
[TAB]      .PSECT [TAB] DATA,D,RW [RET]
[TAB][RET]
; [RET]
; MODULE FUNCTION DETAILED: [RET]
[TAB][RET]
; [RET]
; [TAB]    INPUTS: [RET]
; [RET]
; [TAB]    OUTPUTS: [RET]
; [RET]
; [TAB]    SIDE EFFECTS: [RET]
; [RET]
[TAB][RET]
[TAB]      .PAGE [RET]
[TAB]      .SBTTL [RET]
[TAB]      .PSECT [RET]
```

(Continued on next page)

Example 2–2 (Cont.): Creating the Skeleton File SKEL.MAC

```
; START CODE HERE [RET]
START: [RET]
[TAB][RET]
END: [TAB]  EXIT$S [TAB] [TAB] [TAB]        ;EXIT CLEANLY TO EXEC [RET]
[TAB]       .END [TAB] [TAB] [TAB]          ;TELL ASSEMBLER END OF CODE [RET]
[RET]
[GOLD][COMMAND]
Command: EXIT [ENTER]
$
```

2.2.4 Creating a Source File from the Skeleton

After you create the skeleton file, you can use it many times to create different source files by running the EDT editor again as described in Section 2.2.1.

Example

```
$ EDIT [RET]
File? SKEL.MAC [RET]
    1
*c [RET]
```

Finds the file you just created, reads it into memory, and enables keypad editing at the beginning of the file.

The EXIT command with a file specification creates a new file having that name and containing all of the text in your skeleton. For example, you can first press the GOLD key at the prompt and then press the COMMAND function key, as follows:

```
[GOLD] [COMMAND]
Command: EXIT FILE.MAC [RET]
```

EDT creates either the new file FILE.MAC;1 in your directory or, if the file already exists, a new version of the file. It retains the input file SKEL.MAC. You can repeat this process to create as many new source files as you need.

At this point, the contents of SKEL.MAC and your new file are exactly the same—typographical errors and all. Now you must use editing commands to change your new file to make it unique.

By using the same skeleton file each time you want to create a new source file, you save typing time and have a better chance of creating consistent, easily readable, and well-documented programs. Use EDT and the skeleton file (SKEL.MAC) to create the example source modules shown in Examples 2–3, 2–4, and 2–5. These modules are used in Chapters 3 and 4.

The MicroPDP-11 instructions used in the programming examples in this manual are listed in the *PDP-11 Programming Card.*

Example 2–3: Source Code for FILE.MAC

```
        .TITLE  NUMA    COUNT NUMBER OF A'S
        .IDENT  /01/            ; IDENTIFY MODULE VERSION
;
;
; AUTHOR: Z
;

;
; CHANGES:
;

;
; MODULE FUNCTION GENERAL:
;       THIS MODULE LOADS BUFFER,
;       COUNTS THE NUMBER OF A'S (UPPER
;       CASE ONLY) IN THE BUFFER, CONVERTS
;       THE NUMBER TO OCTAL, AND REPORTS
;       THE NUMBER OF A'S FOUND.
;
        .PAGE                   ; BREAK PAGE FOR PREFACE
        .SBTTL  SYMBOL, MACRO, DATA DEFINITIONS
        .LIST   TTM             ; TERMINAL LISTING MODE
        .NLIST  BEX             ; SUPPRESS BIN EXTENSION
        .MCALL  EXIT$S          ; EXEC'S EXIT MACRO

;
; LOCAL SYMBOL DEFINITIONS:
        MSGLEN  = NUMEND-MSG
        SIZ     = 80.
        SIZA    = 6.
;
;
; LOCAL MACROS: NONE
;

;
; LOCAL DATA BLOCKS:
;
        .PSECT  DATA,D,RW
A:      .ASCII  /A/             ; DEFINE AN A
BUF1:   .BLKB   SIZ             ; DEFINE BUFFER
MSG:    .ASCII  /THE NUMBER OF A'S IS /
NUMA:   .BLKB   SIZA            ; DEFINE OCTAL COUNT
NUMEND  = .                     ; END OF MESSAGE
        .EVEN
NUMC:   .BLKW   1               ; NUMBER OF CHARS TYPED
;
;
; MODULE FUNCTION DETAILED:
;       INPUTS:
;               BUF1 IS LOADED WITH CHARACTERS
;
```

(Continued on next page)

Example 2–3 (Cont.): Source Code for FILE.MAC

```
;       OUTPUTS:
;               NUMA HOLDS THE NUMBER OF A'S
;
;       SIDE EFFECTS: NONE
;
; START CODE HERE

        .PAGE
        .SBTTL  ROUTINE TO COUNT A'S
        .PSECT
START:
        MOV     #BUF1,R0        ; LOAD BUFFER ADDR
        MOV     #SIZ,R1         ; LOAD BUFFER SIZE
        CALL    READ            ; READ FROM TTY
        TST     R2              ; ANY CHARS IN BUFFER?
        BEQ     END             ; IF NONE, FINISH UP
        CLR     R1              ; INIT # OF A'S COUNTER
        MOV     R2,NUMC         ; SAVE # OF CHARS TYPED
10$:
        CMPB    (R0)+,A         ; IS CHAR = A?
        BNE     20$             ; IF NO, BET NEXT CHAR
        INC     R1              ; COUNT AN A
20$:
        DEC     R2              ; ONE LESS CHAR
        BNE     10$             ; IF MORE, COMPARE NEXT
        .PAGE
        .SBTTL  TRANSLATE COUNT TO OCTAL
        MOV     #NUMA+6,R0      ; SET PTR TO OCTAL #
        MOV     #5,R2           ; SET COUNT OF DIGITS
30$:
        MOV     R1,-(SP)        ; STACK IS TEMP AREA
        BIC     #177770,@SP     ; STRIP LOW 3 BITS
        ADD     #60,@SP         ; MAKE OCTAL DIGIT
        MOVB    (SP)+,-(R0)     ; STORE OCTAL DIGIT
        ASR     R1              ; SHIFT TO
        ASR     R1              ;       NEXT
        ASR     R1              ;                3 BITS
        DEC     R2              ; ONE LESS DIGIT
        BNE     30$             ; IF MORE, REPEAT
        MOV     #MSG,R0         ; LOAD ADDR OF BUFFER
        MOV     #MSGLEN,R1      ; LOAD SIZ OF MESSAGE
        CALL    WRITE           ; REPORT THE RESULTS
END:    EXIT$S                  ; EXIT CLEANLY TO EXEC
        .END                    ; TELL ASSEMBLER END OF CODE
```

After you have typed in the code, use the techniques described in Sections 2.2.1 to 2.2.3 to create two new source files, FILEA.MAC and FILEB.MAC, from the skeleton file. The code for these two files is shown in Examples 2–4 and 2–5. These two files and the file FILE.MAC will be used in Chapter 4 to build and test a task. You may want to edit the skeleton file before you create the two new source files.

Example 2–4: Source Code for FILEA.MAC

```
        .TITLE  TTREAD  TERMINAL READ SUBROUTINE
        .IDENT  /01/

;
; AUTHOR: DEF   8-AUG-86
;

;
; CHANGES: NONE
;

;
; MODULE FUNCTION GENERAL:
;       THIS MODULE READS A LINE FROM A
;       TERMINAL INTO A BUFFER

;
        .PAGE                   ; BREAK PAGE FOR PREFACE
        .SBTTL  SYMBOL, MACRO, DATA DEFINITIONS
        .LIST   TTM             ; TERMINAL LISTING MODE
        .NLIST  BEX             ; SUPPRESS BIN EXTENSION
        .MCALL  QIO$S,WTSE$S

;
; LOCAL SYMBOL DEFINITIONS:
        EFN1    = 1
        LUN5    = 5
;

;
; LOCAL MACROS: NONE
;

;
; LOCAL DATA BLOCKS:
;
        .PSECT  DATA,D,RW
IOST:   .BLKW   2               ; DEF IO STATUS WDS
;
;
; MODULE FUNCTION DETAILED:
;       INPUTS: R0 = ADDRESS OF BUFFER TO LOAD
;               R1 = LENGTH IN BYTES OF BUFFER
;
;       OUTPUTS:        R2 = NUMBER OF CHARS (BYTES) READ
;
;       SIDE EFFECTS:   IOT IF ERROR
;
        .PAGE
        .SBTTL  START OF CODE
        .PSECT
```

(Continued on next page)

Example 2–4 (Cont.): Source Code for FILEA.MAC

```
; START CODE HERE:
READ::                          ; DEFINE ENTRY POINT
        QIO$S   #IO.RLB,#LUN5,#EFN1,,#IOST,,<R0,R1>
                                ; QIO DIR PARAMETERS:
                                ;       RLB IS READ LOG BLOCK
                                ;       LUN5 = TKB DEFAULT
                                ;       EFN1 IS EVENT FLAG #1
                                ;       IOST = STATUS AREA
                                ;       <> = PARAMETER LIST
                                ;         R0 = START OF BUFFER
                                ;         R1 = SIZE OF BUFFER
        BCS     10$             ; IF SET, DIR ACCEPT ERROR
        WTSE$S  #EFN1           ; WAIT FOR IO COMPLETE,EF 1
        TSTB    IOST            ; CHECK IO STATUS
        BLT     10$             ; IF LT, IO ERROR
        MOV     IOST+2,R2       ; SAVE # OF BYTES READ
        RETURN                  ; GO BACK TO CALLER
10$:
        MOV     $DSW,R0         ; SAVE DIR STAT WD
        MOVB    IOST,R1         ; SAVE IO STAT BYTE
        IOT                     ; FORCE SST EXIT
        .END                    ; TELL ASSEMBLER END OF CODE
```

Example 2–5: Source Code for FILEB.MAC

```
        .TITLE  TTWRIT  TERMINAL WRITE SUBROUTINE
        .IDENT  /01/

;
; AUTHOR: DEF 8-AUG-86
;

;
; CHANGES: NONE
;

;
; MODULE FUNCTION GENERAL:
;
;       THIS MODULE WRITES A
;       LINE FROM A BUFFER TO
;       A TERMINAL
;
        .PAGE                   ; BREAK PAGE FOR PREFACE
        .SBTTL  SYMBOL, MACRO, DATA DEFINITIONS
        .LIST   TTM             ; TERMINAL LISTING MODE
        .NLIST  BEX             ; SUPPRESS BIN EXTENSION
        .MCALL  QIO$S,WTSE$S

;
; LOCAL SYMBOL DEFINITIONS:
        EFN1    = 1
        LUN5    = 5
;

;
; LOCAL MACROS: NONE
;

;
; LOCAL DATA BLOCKS:
;
        .PSECT  DATA,D,RW
IOST:   .BLKW   2               ; DEF IO STATUS WDS
;
; MODULE FUNCTION DETAILED:

;       INPUTS:
;
;               R0 = ADDR OF BUFFER TO WRITE
;               R1 = LENGTH IN BYTES OF BUFFER
;       OUTPUTS:
;
;               SUCCESS IN IOST
;       SIDE EFFECTS: IOT IF ERROR
```

(Continued on next page)

Example 2–5 (Cont.): Source Code for FILEB.MAC

```
;
        .PAGE
        .SBTTL  START OF CODE
        .PSECT
; START CODE HERE
WRITE::                         ; DEF ENTRY POINT
        QIO$S   #IO.WLB,#LUN5,#EFN1,,#IOST,,<R0,R1,#40>
                                ; QIO$S PARAMETERS:
                                ;  IO.WLB FUNCTION CODE
                                ;  LUN5 (TKB DEFAULT)
                                ;  EFN1 IS EVENT FLAG 1
                                ;  STATUS AREA = IOST
                                ;  PARAMETER LIST <>
                                ;   R0 = START OF BUFFER
                                ;   R1 = # OF CHARS TO WRITE
                                ;   40 = OUTPUT <CR>,<LF>
        BCS     10$             ; IF SET, DIR ACCEPT ERROR
        WTSE$S  #EFN1           ; WAIT FOR IO COMPLETE
        TSTB    IOST            ; CHECK IO STATUS
        BLT     10$             ; IF LT, IO ERROR
        RETURN                  ; GO BACK TO CALLER
10$:
        MOV     $DSW,R0         ; SAVE DIR STAT WD
        MOVB    IOST,R1         ; SAVE IO STAT VALUE
        IOT                     ; SST DUMPS TASK REGS
        .END                    ; TELL ASSEMBLER END OF CODE
```

Chapter 3
Assembling and Correcting a Program Module

This chapter describes uses of the MACRO-11 assembler, common types of coding errors, ways to uncover and correct errors, and the way to generate a cross-reference listing. For a detailed description of the options available while assembling a program, refer to Appendix A for an explanation of the MACRO command.

The material in this chapter assumes that you have created the three source files described in Chapter 2.

3.1 Performing a Diagnostic Run on a Source File

Your first use of the MACRO-11 assembler on a source file should be to perform a diagnostic run. You run the assembler only to check for general errors, not to produce an object module or a listing file.

To perform a diagnostic run from a DCL terminal, type the following command line:

```
$ MACRO/NOOBJECT/DISABLE:GLOBAL FILE [RET]
     (any error messages appear)
$
```

The source file is named FILE.MAC, but because the MACRO command defaults to a file type of MAC, the file type need not be specified. Normally, the MACRO command is used to create an object module, making the /NOOBJECT qualifier necessary if you want to override this standard function. The MACRO command does not produce a listing file unless you request one with the /LIST qualifier. When you do not make a listing file, any errors that result from assembly are listed directly on your terminal.

The /DISABLE:GLOBAL qualifier causes the MACRO-11 assembler to disable the setting of undefined symbols to global and external. Ordinarily, when the MACRO-11 assembler finds a symbol that is not defined in the source file, it assumes that the reference is to a symbol defined externally, that is, in another module. By disabling this feature for your diagnostic run, you tell the assembler to flag any potential global references as an undefined symbol error. Because you already know which symbols in your source file are global, this disabling method is a convenient way to catch typographical errors in other symbol names.

The appearance of MACRO-11 messages at the terminal during the diagnostic run indicates that your module contains errors. If the assembler does not find any errors, it simply returns control to the Executive, and DCL prints the dollar sign prompt ($). Errors in the assembly are denoted by single-letter codes printed at the beginning of the faulty statement. These errors are summarized in the *PDP-11 MACRO-11 Language Reference Manual.*

The only errors that should appear from the diagnostic run are the following:

```
U    71 000010  004767          CALL      READ      ; READ FROM TTY
U    99 000110  004767          CALL      WRITE     ; REPORT THE RESULTS
ERRORS DETECTED:  2
/DS:GBL=FILE
```

The two undefined symbols READ and WRITE are the entry points defined in the source files FILEA.MAC and FILEB.MAC. These symbols are to be resolved by TKB. (Note that this example was generated by the command MACRO/DISABLE:GLOBAL FILE.)

3.2 Errors Encountered During Assembly

Four error codes cover the majority of errors made in an assembly language source file. The following sections describe some of the most common conditions that generate these error codes.

3.2.1 MACRO-11 Error Code A

Error code A indicates a general assembly error. Most of these errors are caused by typing mistakes such as the following:

- Omitting the semicolon (;) from a comment

  The semicolon separates your comment from the portion of the statement that the assembler evaluates. If you omit the semicolon, MACRO-11 attempts to evaluate your comment as part of the rest of the statement line.

- Omitting the period (.) from a MACRO-11 directive

  The leading period in the operator field tells the assembler that the statement contains a MACRO-11 directive. If you forget to include the period on a directive, the assembler cannot evaluate the operator as a directive. As a result, error code A is generated, the directive and its arguments are given a value of 0, and they are designated as global symbols.

- Misspelling a PDP-11 instruction mnemonic

  If you misspelled a PDP-11 instruction mnemonic (for example, MOVE instead of MOV), the assembler can evaluate the operands but not the operator. The *PDP-11 MACRO-11 Language Reference Manual* lists all the mnemonics alphabetically. These mnemonics make up the permanent symbol table (PST). The *PDP-11 Programming Card* also contains all th instruction mnemonics.

- Forming an illegal symbol

  The first character of a symbol must not be a numeral.

- Delimiting a directive argument improperly

  Many MACRO-11 directives require a character or argument string to begin and end with a certain delimiting character. If you use the wrong character, or omit one of the delimiters, the assembler cannot properly match the delimiters and therefore cannot evaluate the directive. For example, the .ASCII directive requires the character string to begin and end with the same delimiting character.

Another type of general assembly error involves addressing errors. The typical addressing error occurs when you exceed the range of a branch instruction (that is, branching more than 128 words backward or 127 words forward). To correct this type of error, replace the branch instruction with the following:

- Code that tests the proper condition
- The JMP instruction (to transfer control)

Illegal forward references are also common as general assembly errors. If you define a symbol based on another symbol that is defined by a forward reference, the assembler cannot evaluate the reference. Note the following example:

```
A = B + 10.
C = A + 10.
```

The assembler cannot evaluate the symbol A because B is not yet defined.

3.2.2 MACRO-11 Error Code U

Error code U signals an undefined symbol error. This error usually occurs for one of the following reasons:

- A symbol name on the .MCALL directive was misspelled.
- A reference was made to a local label that does not exist in the current local symbol block.

3.2.3 MACRO-11 Error Code Q

Error code Q indicates questionable syntax. This error usually results from one of the following:

- Including too many (or too few) arguments in a directive
- Specifying an incorrect number of operands in an instruction
- Omitting the semicolon from a comment, causing the assembler to attempt to evaluate the comment as part of the statement

3.2.4 MACRO-11 Error Code E

Error code E means that you have omitted the .END directive from the assembly language source file. If the assembler does not find the .END directive, it generates error code E with a line number of 0 after the last statement in the listing file.

Error code E also may indicate an expression overflow. If the assembler encounters a nested expression that is too complex, it generates error code E and denotes the point of the overflow with a question mark (?). To clear the error condition, either simplify the expression or ask your system manager to build the MACRO-11 assembler with a larger stack.

3.3 Generating a Program Module and a Listing

After you correct the errors uncovered in the diagnostic run, you are ready to produce an object module and a listing file. The following DCL command line produces an object module and a listing file:

```
$ MACRO FILE/LIST [RET]

     (error summary printed)

$
```

This command line, like the command line for the diagnostic run, assumes default file types for the object file and the listing file. The assembler creates an object module called FILE.OBJ. The /LIST qualifier causes the assembler to create a file called FILE.LST. It is good programming practice to use the assembler defaults for file types and file names. Using the defaults helps you to differentiate file types and to group associated files under the same name. If you wish to use other file names and file types, you can override the defaults by supplying complete file specifications as arguments to the /LIST and /OBJECT qualifiers.

Note that the /LIST qualifier is added to the file specification rather than to the MACRO command in the example. This placement of the qualifier causes a listing file to be created in your directory, but the file is not printed on the line printer. The following MACRO command line causes the listing file FILE.LST to be created in your directory, but the file is also printed on your system's line printer:

```
$ MACRO/LIST FILE [RET]

     (error summary printed)

$
```

For the time being, you should use the /LIST qualifier as a file specification qualifier, to keep from printing too many copies. During the program development cycle, you create many files for which you do not need a permanent copy. It is easier and less wasteful to examine a listing file at your terminal than to generate numerous printed copies of listing files that must be discarded because of minor errors. After you attain an error-free assembly, you can print a copy of the latest version of the listing file.

When you request a listing file, the errors are printed in the file, not on your terminal. All you see on your terminal is a message giving the total number of errors found. If no message appears, there are no errors. Note, however, that freedom from assembly errors does not guarantee that the program will run properly.

You can issue the following command lines to assemble the two other source files, FILEA.MAC and FILEB.MAC, which you created using the procedures described in Chapter 2:

```
$ MACRO  FILEA/LIST [RET]
$ MACRO  FILEB/LIST [RET]
```

These two command lines create the object modules, FILEA.OBJ and FILEB.OBJ, that you will need to link into your task in Chapter 4.

3.4 Examining a Listing at the Terminal

Use the TYPE command to display the listing file at your terminal, as shown:

```
$ TYPE FILE.LST [RET]

     (file appears on screen)

$
```

You can use control commands to temporarily stop and restart the display, and to alternately suppress and resume the output request. The commands are summarized in Table 3–1.

Table 3–1: Terminal Output Control Commands

Command	Effect
CTRL/S	Temporarily stops the display.
CTRL/Q	Restarts the display stopped by CTRL/S.
CTRL/O	Alternately suppresses and resumes the output to the terminal.

The CTRL/S and CTRL/Q commands are used together to freeze the display on the screen and to request more lines to be displayed. While the CTRL/S command is in effect, you can read what is on the screen. The CTRL/Q command directs the system to restart the display where it left off when it detected the CTRL/S command. The NO SCROLL or HOLD SCREEN key can be used to accomplish the same thing.

The CTRL/O command is used to suppress unwanted output. The command directs the system to stop sending characters to the terminal. The program, however, continues processing but simply omits displaying the output. (While the CTRL/O command is in effect, the system disables keyboard input and does not echo any characters typed at the terminal.) By typing CTRL/O again, you direct the system to resume output to the terminal. By typing successive CTRL/Os, you can skip unnecessary portions of the output until the program reaches the correct part. If the program finishes processing the output request while the CTRL/O command is in effect, the system automatically reenables keyboard input and the dollar sign prompt ($) appears on the terminal.

3.5 Generating a Cross-Reference Listing

Symbol and macro cross-reference listings are worthwhile additions to the assembly listing. These listings give, in alphabetical order, each symbol and macro name defined or referred to and the number of the page and line in the listing where the definition or reference occurs. Type the following command line to obtain cross-reference listings:

```
$ MACRO/NOOBJECT FILE/CROSS_REFERENCE [RET]

     (any errors cause total number to be printed)

$
```

Note that the command line does not include the /LIST qualifier. The /CROSS_REFERENCE qualifier implies the /LIST qualifier because the cross-reference listing is attached to the assembly listing. If you want to have the listing printed, use the /CROSS_REFERENCE qualifier as a command qualifier instead of as a file qualifier.

Remember, you do not need to type the full form of the command unless you are keeping a record of terminal activity. The following command line has the same effect as the previous one:

```
$ MAC/NOOB FILE/CRO [RET]
```

The CRF task appends the cross-reference listing to the end of the listing file, denoting the cross-references by the titles SYMBOL CROSS REFERENCE and MACRO CROSS REFERENCE.

3.6 Printing a Copy of Listings

Once you have developed an error-free assembly, you can obtain a hard copy of the listing for reference. From your terminal, type the following command line:

```
$ PRINT FILE.LST [RET]
$
```

The command line creates a request to the spooling task to print the file you specify. (You can specify more than one file at a time by listing more than one file specification in the command line, separating each with a comma.) Your request is placed in a queue with other requests.

3.7 Cleaning Up the Disk Directory

After you edit and reassemble the source files several times, your directory becomes cluttered with multiple versions of the same files. A DIRECTORY command is provided for listing information about files. The following command lists all the files in your directory:

```
$ DIRECTORY [RET]
     (the directory listing appears)
$
```

You will notice in the directory a number of files with the same file name and file type, but different version numbers. Use this command to purge all but the most recent version of these files as follows:

```
$ PURGE *.MAC,*.LST,*.OBJ [RET]
```

Chapter 4
Building and Testing a Task

This chapter describes ways to use the Task Builder (TKB) to create a task image from program object modules. The procedures described in this chapter assume that you have created three error-free object modules, as described in Chapter 3.

4.1 Creating a Task Image

TKB creates a task image file that can be loaded into memory. You can supply as input to TKB either a single object module or multiple object modules. In most cases, however, your programs will consist of multiple object modules. The following sections describe the procedures and the way TKB reports error conditions.

4.1.1 Supplying a Single Object Module

Use the DCL command LINK to invoke TKB and create a task image file from a single object module, as follows:

```
$ LINK FILE [RET]
     (any error messages appear)
$
```

Once again, all defaults are applied automatically. The LINK command defaults to an object module in a file named FILE.OBJ and causes TKB to produce a file named FILE.TSK containing the task image.

TKB tries to resolve all global references in the object module. If there are undefined references after the module has been processed, TKB searches the system object library SYSLIB.OLB in system directory [1,1] on the library device (LB). If no errors are encountered in the process, TKB exits and the dollar sign prompt ($) appears.

If TKB detects an error during processing, it prints a message in one of the following forms:

```
LINK -- *DIAG* - error message
```

or

```
LINK -- *FATAL* - error message
```

TKB error messages are summarized in the *RSX-11M-PLUS and Micro/RSX Task Builder Manual.*

If an error message appears and the error condition described is not operational (for example, lack of space for the task image file) or is not a fatal error, TKB creates the task image file anyway. Depending on the error condition, you may have to remove the cause of the error from the source file, reassemble the source file, and repeat the TKB procedure. In some instances, the diagnostic condition is merely a warning and has no ill effect when the task runs. (For guidelines on correcting error conditions, see Section 4.4.)

When you create the task image from the single object module FILE.OBJ, TKB prints the following error message:

```
TKB -- *DIAG-*2 Undefined symbols segment FILE

READ
WRITE
```

The undefined symbols READ and WRITE are the entry points of the two routines defined by the object modules FILEA.OBJ and FILEB.OBJ. TKB searches the system object library to resolve global references left undefined in your input. Because TKB failed to find modules that defined these symbols, it reported the error condition. You can eliminate the error condition by following the procedures described in Section 4.1.2.

4.1.2 Supplying Multiple Object Modules

TKB accepts multiple object modules as input to the LINK command. At your terminal, type the names of the object files, separated by commas, as shown:

```
$ LINK FILE,FILEA,FILEB [RET]
     (any error messages appear)
$
```

The LINK command defaults to the file type OBJ for the three input files. The resulting task image file is named FILE.TSK. The LINK command defaults to the name of the first object file named to derive the name of the TSK file.

TKB performs the same actions as those described in Section 4.1.1 for one object module. Only one of the object modules specified must have been assembled with a .END directive that gives the starting address of the task. If none of the modules contains the starting address, TKB assigns the default transfer address of 1, which causes an error when you run the task. Refer to Section 4.4 for more information on running the task and correcting errors.

TKB can also use a concatenated object module as input, which is merely a file containing multiple object modules. To create a concatenated file, use the DCL command COPY, as shown:

```
$ COPY [RET]
From? FILE.OBJ,FILEA,FILEB [RET]
To? FILCON.OBJ [RET]
$
```

or

```
$ COPY FILE.OBJ,FILEA,FILEB FILCON.OBJ [RET]
$
```

The response to the From? prompt lists the files to be concatenated. Note that you need specify the file type only on the first file listed. This file type becomes the default file type for subsequent files. The COPY command automatically concatenates these files into a single output file.

The single concatenated object file can then be the sole input to the LINK command, as shown in the following command line:

```
$ LINK/TASK:FILE FILCON [RET]
     (any error messages appear)
$
```

This operation saves file processing overhead for TKB. As a result, building a task from a concatenated file is possibly 40 percent faster than listing the object modules separately.

4.2 Task Builder Defaults

When you build a task image, TKB applies certain default conditions to your program, including the partition in which your task runs, the host system memory management characteristics, the task's checkpointability, and the number of logical units your task can access. If your program does not use the default conditions, the process of building a task becomes more complex. You can consult the *RSX-11M-PLUS and Micro/RSX Task Builder Manual* for the procedures that override the default conditions.

TKB assigns your program to be run in the default partition, called GEN. If you are building a task to run in another partition, you can either supply the correct partition name at run time or rebuild the task and specify the correct partition name then (refer to the description of the PAR option in the *RSX-11M-PLUS and Micro/RSX Task Builder Manual*).

Using TKB memory management characteristics, TKB allocates memory starting at virtual address 0 and assumes that the task will be relocated by memory management hardware. As a result, the task can be run in any partition large enough to contain the image. TKB also assumes that the task is not checkpointable and, therefore, uses the Floating Point Processor.

TKB establishes the maximum number of logical units (six) the task can access and supplies the assignments for these logical units. The default assignments are the following: logical units 1 to 4 are assigned to the system device (SY); unit 5 is the task-initiating terminal (TI); and unit 6 is the console listing device (CL).

4.3 Generating a Map and a Global Cross-Reference Listing

Before you run the task and correct simple errors, you can produce a memory allocation file (called a map) and a cross-reference listing of global symbols. The map and global cross-reference file are useful in later stages of program development and for program documentation.

4.3.1 Requesting a Map and a Global Cross-Reference Listing

In most situations, you need a standard map and a global cross-reference listing for debugging a task. To create a map with a global cross-reference listing, type the following command line:

```
$ LINK/CROSS_REFERENCE/NOWIDE/NOTASK FILE,FILEA,FILEB [RET]
$
```

The /NOTASK qualifier suppresses the creation of a task image file. You request a cross-reference listing with the /CROSS_REFERENCE qualifier. The /NOWIDE qualifier reduces the width of the listing from 132 columns to 80 columns for display on a terminal. Because /CROSS_REFERENCE implies a map, you do not have to specify the /MAP qualifier.

If you wish to create both the task image file and the map with the cross-reference listing at the same time, use the following command line:

```
$ LINK/CROSS/NOWIDE FILE,FILEA,FILEB [RET]
$
```

TKB creates both FILE.TSK and FILE.MAP. The map includes a cross-reference listing.

4.3.2 Examining the Map at the Terminal

Use the TYPE command to examine the map at your terminal. The command line is as follows:

```
$ TYPE FILE.MAP [RET]

     (file appears on screen)

$
```

Use the control commands CTRL/S, CTRL/Q (or NO SCROLL and HOLD SCREEN), and CTRL/O (all summarized in Table 3-1) to control the terminal output.

4.3.3 Requesting a Full Map

The map file produced, as described in Section 4.3.1, is a short form of the map that contains most of the information needed for debugging tasks. To generate a full form of the map, use the following command line:

```
$ LINK/LONG/MAP:FULL/CRO/SYSTEM_LIBRARY_DISPLAY/NOTASK FILE,FILEA,FILE [RET]
$
```

The /LONG qualifier indicates that you want the long form of the map and causes TKB to add a file contents section to the map. The /LONG qualifier implies the /MAP qualifier; however, the /MAP qualifier is used here to give the map file the name FULL.MAP so you can distinguish the different maps you have made during this demonstration session. The /CRO qualifier is the abbreviated form of the /CROSS_REFERENCE qualifier. The /SYSTEM_LIBRARY_DISPLAY qualifier (usually abbreviated /SYS) instructs TKB to include system library contributions to the task in the file contents section of the map. (System symbols are also included in the global cross-reference listing.)

4.4 Running the Task and Correcting Typical Errors

You execute your task by using the RUN command and the name of the task image file.

Example

```
$ RUN FILE RET
```

Because the task FILE is not installed on the system, the RUN command searches your User File Directory (UFD) on device SY for a file named FILE.TSK. RUN, installs it temporarily, and runs it immediately. (The task will be automatically removed when the task exits.) To run task FILE, the Executive transfers control to the task starting, or transfer, address. If your task encounters an error condition, the Executive must decide whether to abort the task.

Errors that can cause the Executive to abort a task are either hardware related or software related. If the error is hardware related, such as a memory parity error or a load failure, the Executive begins aborting the task. In contrast, a synchronous system trap (SST) error condition (software related), such as an illegal instruction, causes the Executive to attempt to transfer control to an SST routine. An SST routine is a routine within a task that services a particular type of SST condition. If your task defines a routine to service the type of trap, the Executive transfers control to it. If your task does not have the routine defined, the Executive aborts the task.

Aborting a task forces an orderly termination of the task. Included in the termination is a request for the Task Termination and Notification task (TKTN) to display a message on your terminal. The program display includes the cause of the abort and a list of the task registers and Processor Status Word (PSW), for example:

```
    14:16:26  Task   "TT30  " terminated
              Reserved inst execution
              R0=001401
              R1=100076
              R2=000000
              R3=140130
              R4=000000
              R5=000000
              SP=001254
              PC=001262
              PS=170001
$
```

The information can help you ascertain the cause of the abort. If the cause of the error is hardware related, report the occurrence to your system manager, who can consult the error-logging data to find where the problem originated. If the cause of the error was an SST (software related) condition, you can use the data displayed by TKTN to find the problem.

The value of the program counter (PC), minus 2, shown in the display tells you the address of the instruction that was being executed when the error was encountered. In the example shown above, the PC is at the address 001262.

To correct any errors in your task, you must edit the source file or files concerned, reassemble the corrected files, and rebuild the task.

Chapter 5
Using Debugging Aids

This chapter introduces the following three debugging aids that are helpful in the program development process:

- The On-Line Debugging Tool (ODT)
- The Postmortem Dump (PMD)
- The snapshot dump ($SNAP)

There is also another debugging tool called the Executive Debugging Tool (XDT) that is used for debugging privileged code. Refer to the *RSX-11M-PLUS and Micro/RSX XDT Reference Manual* for a complete description of XDT.

5.1 Using the On-Line Debugging Tool

The On-Line Debugging Tool (ODT) is special code that you include in your task image to assist you during debugging. ODT gives you interactive control of task execution, and allows you to set breakpoints and to examine and change data or instructions within the memory-resident task. The ODT module is linked into your task image, thereby increasing the size of the task image. Therefore, you remove ODT from your task when you finish debugging, by rebuilding the task and omitting the ODT module.

ODT commands differ from commands in other utility programs. Most programs have multicharacter commands that require a line terminator before they are executed. ODT commands, however, are single characters and require no line terminator. That is, ODT interprets input on a character-by-character basis rather than on a line-by-line basis. Therefore, as soon as you type a character that ODT recognizes as a command, ODT interprets it and performs the specified function. This difference in commands means that you must be careful when you are debugging your task with ODT.

5.1.1 Including ODT in a Task

The /DEBUG qualifier to the LINK command is used to include ODT in a task. An example follows:

```
$ LINK/DEBUG/TASK:BUG/MAP:BUG/CROSS_REFERENCE FILE,FILEA,FILEB [RET]
$
```

The /DEBUG qualifier specifies that you want to include ODT in the task. The /TASK:BUG qualifier specifies that you want the task image file to be named BUG.TSK. The /MAP:BUG qualifier specifies that you want the map to be named BUG.MAP. In this way, you can tell the difference between the versions of the task-built file with ODT and without. The Task Builder (TKB) accesses the file LB:[1,1]ODT.OBJ and links it into the task. The /CROSS_REFERENCE qualifier implies a /MAP qualifier. An accurate map of the task is necessary for use with ODT.

On systems that support instruction and data space, the file LB:[1,1]ODTID.OBJ will be referenced. If the task is built with separate instruction and data space, the file LB:[1,1]ODTID.OBJ will be inserted in the task.

5.1.2 Preparing to Use ODT

Before you run a task containing ODT, ensure that accurate listings of the assembled source files are available. These listings show the offsets into the modules in your task. The map of the task and the assembled source listings provide the data you need to set breakpoints and examine locations within the task.

5.1.3 Setting Up the Task

When you run a task containing ODT, ODT gains control, identifies itself (and the task it controls), and prints its command prompt. The following lines show the sequence:

```
$ RUN BUG [RET]
ODT:TT5
-
```

The notation TT5 is the name that the system dispatcher assigned to the task. Such a name consists of the letters TT followed by the unit number of the terminal that requested the task. (The task shown here was run from terminal number 5_8.)

The underline character (_) is ODT's prompt. It indicates that ODT is ready to accept commands.

5.1.4 Relocation Registers

To access locations within the task, you should establish one or more relocation registers. This set of eight registers, numbered $R0 to $R7, allows you to specify locations within the task in terms of offsets from the start of modules in the task image.

To establish the proper addressing using offsets, you must first consult the location information in the task map. On the map listing, the portion titled Memory Allocation Synopsis contains the location information for each program section and for each contribution to the program sections from different modules. A sample of the relevant portion of the map for the program BUG is shown in Example 5–1.

Example 5-1: Memory Allocation Synopsis from Task BUG Map

```
Memory allocation synopsis:
Section                                          Title  Ident  File
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON)      001202 000340 00224.
                               001202 000122 00082. NUMA   01     FILCON.OBJ;1
                               001324 000110 00072. TTREAD 01     FILCON.OBJ;1
                               001434 000106 00070. TTWRIT 01     FILCON.OBJ;1
DATA  :(RW,D,LCL,REL,CON)      001542 000166 00118.
                               001542 000156 00110. NUMA   01     FILCON.OBJ;1
                               001720 000004 00004. TTREAD 01     FILCON.OBJ;1
                               001724 000004 00004. TTWRIT 01     FILCON.OBJ;1
$$$ODT:(RW,I,GBL,REL,OVR)      001730 005654 02988.
                               001730 005654 02988. ODTRSX M06    ODT.OBJ;121
```

The location information for a program section is the octal starting address of the program section and its extent in bytes (both octal and decimal values). For example, for the blank program section, the starting location is 1202_8 and the extent is 340_8, or 224_{10}, bytes. The octal starting addresses and extents (in bytes) for the contributions from each object module are listed under the program section location information. For example, the contribution from TTREAD in the blank program section starts at location 1324 and extends for 110_8, or 72_{10}, bytes.

The following example shows how to place the starting addresses of the modules in relocation registers:

```
_1202;0R
_1324;1R
_1434;2R
_1542;3R
_1720;4R
_1724;5R
```

The R commands place the addresses in relocation registers 0 to 5. (The addresses are octal; ODT accepts only octal numbers.) As soon as you type the R in the command line, ODT generates line-feed and carriage return operations and prints another prompt. This action indicates that ODT has executed the command as soon as it was typed. Therefore, before typing the R (or any command), make sure that the command line is correct.

If you notice a typographical error in the line before you type the command itself, simply type CTRL/U, type the number 8 or 9, or press the DELETE key, as shown in the following example:

```
_1272;08?
-
```

ODT considers the decimal number 8 an illegal character. It discards the input line, displays a question mark (?) to signal an error, and prints the prompt on a new line. You must retype the entire line. If you do enter an incorrect address in the relocation register, simply retype the command, as follows:

```
_1272;0R
_1202;0R
```

ODT stores the most recently entered value in the register. To access a location within a task most conveniently, you must create an address made up of the values stored in the relocation register and a value showing the distance of the location from the relocation register value.

The relocation register provides the base address of a module; the location counter value supplies an offset to the location within the program section for the module. The command 1202;0R places the starting address of the NUMA contribution to the blank program section in relocation register 0. Location counter value 20 in the assembly listing for NUMA is 20 bytes from the start of the address in relocation register 0. You use the two values to form the address of the location. The address is formed by typing the number of the relocation register, a comma (,), and the octal offset value. For example:

```
0,20
```

ODT adds the base value in relocation register 0 (1202 in this case) and the offset typed after the comma (20). This creates an effective address of 1222_8. You use this syntax with various ODT commands to access locations within the task address space.

Example 5–2 shows a portion of the assembly listing for the blank program section in the module NUMA.

Example 5–2: Portion of Assembly Listing for NUMA

```
NUMA   COUNT NUMBER OF A'S     MACRO M1200  8-AUG-86 12:39  PAGE 3
ROUTINE TO COUNT A'S

     66                         .SBTTL  ROUTINE TO COUNT A'S
     67 000000                  .PSECT
     68 000000          START:
     69 000000  012700          MOV     #BUF1,R0        ; LOAD BUFFER ADDR
     70 000004  012701          MOV     #SIZ,R1         ; LOAD BUFFER SIZE
     71 000010  004767          CALL    READ            ; READ FROM TTY
     72 000014  005702          TST     R2              ; ANY CHARS IN BUFFER?
     73 000016  001436          BEQ     END             ; IF NONE, FINISH UP
     74 000020  005001          CLR     R1              ; INIT # OF A'S COUNTER
     75 000022  010267          MOV     R2,NUMC         ; SAVE # OF CHARS TYPED
     76 000026          10$:
     77 000026  122067          CMPB    (R0)+,A         ; IS CHAR = A?
     78 000032  001001          BNE     20$             ; IF NO, BET NEXT CHAR
     79 000034  005201          INC     R1              ; COUNT AN A
     80 000036          20$:
     81 000036  005302          DEC     R2              ; ONE LESS CHAR
     82 000040  001372          BNE     10$             ; IF MORE, COMPARE NEXT
```

5.1.5 Examining Locations

To examine words within a module, type the address followed by the slash (/) character, as follows:

```
_0,20/005001
```

The slash character causes ODT to open the designated location as a word and display its contents. To close the currently open location, press either the RETURN key or the LINE FEED key. The RETURN key closes the location, as shown in the following example:

```
_0,20/005001 RET
-
```

ODT closes the location and prints its prompt on a new line.

Once you have opened a location, pressing the LINE FEED key enables you to examine successive words in the task image. The following example shows the procedure:

```
_0,32/001001 [LF]
0,000034 /005201   [RET]
-
```

In response to the LINE FEED key, ODT closes the current location, opens the next sequential location in the task image, and displays the address of the location, a space, the slash character, and the contents of the location. The slash character signals that the location is open as a word.

Note

You can change the contents of the currently open location to n by pressing the octal number n before pressing the RETURN or LINE FEED key. See Section 5.1.7.

To examine bytes within a task instead of words, type the address followed by the backslash (\) character, as follows:

```
_0,32\001
```

The backslash character causes ODT to open the designated location as a byte and display its contents. You can examine successive bytes by pressing the LINE FEED key, after which ODT closes the currently open byte location, opens the next sequential byte location, and displays its contents. For example:

```
_32\001 [LF]
0,000033 \002 [RET]
-
```

The backslash character preceding the contents signals that the location is open as a byte.

Before you proceed in the debugging session, you should verify the relocation register values by examining a location in each module and comparing its contents with the values shown in the assembly listing. The following sequence shows the procedure:

```
_1,66/002403 [RET]
_2,72/000207 [RET]
_3,121\124 [RET]
_4,0/000000 [RET]
_5,0/000000 [RET]
-
```

As you examine each location, compare the contents ODT displays with the assembly listing. If the values do not match, either you have an incorrect listing or the relocation register value is wrong.

5.1.6 Setting Breakpoints Within the Task

To allow you to stop (or break) task execution, ODT provides eight registers called breakpoint registers. These registers, numbered $B0 to $B7, let you specify locations of instructions at which execution should stop.

To establish breakpoints in the task, specify the location of the instruction with the B (BREAK) command shown next.

Format

```
_a;nB
```

Example

```
_0,10;0B
_1,74;1B
-
```

Places the designated addresses in breakpoint registers 0 and 1.

Note

When specifying the address of an instruction, make sure that the location is the first word of the instruction.

As soon as you type the B in the command line, ODT generates the carriage return and line feed operations and prints a prompt. (Changing a breakpoint register is the same as changing a relocation register; simply retype the command line and give the altered contents.)

After setting up the breakpoint registers, you can issue the G (GO) command to begin task execution. For example:

```
_G
0B:0,000010
-
```

When you type the G command, ODT swaps a BPT instruction into each breakpoint location. (The eight breakpoint instruction registers, with register names $I0 to $I7, contain the actual instructions during task execution.) ODT passes control to the starting address of the task. The task executes until it reaches a BPT instruction, at which point ODT regains control. When ODT regains control, the task has not yet executed the instruction at the location where the breakpoint is set. ODT swaps the instructions back into the locations at which breakpoints are set, and it prints a message that supplies the following information:

- The breakpoint register designation
- The relocation address at which execution stopped

In the previous example, the message shows breakpoint register 0 and its contents (offset 10 from the base address in relocation register 0).

5.1.7 Changing the Contents of Locations with ODT

When execution stops at a breakpoint, you can examine and change data within the task image address space. When execution stops at a breakpoint location, the task's general registers are stored in ODT locations accessed by the names $0 to $7. The following sequence shows how to display general registers 0, 1, and 2:

```
_$0/ 001543 [LF]
$1 /000120 [LF]
$2 /135600 [RET]
-
```

The slash (/) character opens the general register as a word location and prints its contents. Pressing the LINE FEED key closes the current location and opens the next sequential location.

To change data, simply type a new value while the current location is open. The following sequence shows how you can change register 2:

```
_$2/ 135600 100 [LF]
$3 /140130 [RET]
-
```

While the location (register 2) is open, you can type the new value to replace the current contents. ODT writes the new value 100_8 into the currently open location before closing it and opening the next sequential location.

Any locations within the task can be examined and changed. The following sequence shows how to open a location as a byte and change its contents:

```
_3,0\101 102 [RET]
_3,0\102 101 [RET]
-
```

The backslash (\) character opens the specified address as a byte location. The new value 102_8 is written to the open location as a byte value. Pressing the RETURN key closes the location. The next command line examines offset 0 to verify that it contains 102_8 and then changes the contents back to 101.

After you examine and change locations, resume execution with the P (PROCEED) command, as follows:

```
_PABCABCABAB [RET]
1B:1,000074
-
```

The P command causes ODT to swap in the BPT instructions, restore the task general registers, and continue with the instruction at which the break occurred.

Note

ODT does not supply a carriage return and line feed after you type the P. Therefore, the data that you type in response to the READ routine will follow the P on the same line.

Execution stops at the location contained in breakpoint register 1.

The G command is used to transfer control to another address and continue execution. For example:

```
_1,76G
```

ODT transfers control to offset 76 and continues execution there. This command purposely tranfers control to the error routine to show what occurs when an error is encountered. See Section 5.1.8.

5.1.8 Error Conditions and Terminating Task Execution

If the task generates an error condition, the Executive handles the processing as a synchronous system trap (SST). Control is passed to ODT, which prints a message similar to the following:

```
IO:2,000000
-
```

This message gives a code describing the reasons for the trap and tells the address following the location that generated the trap. In the previous message, IO means the IOT instruction. If you can discover the cause of the trap, make the appropriate changes in the task and proceed. If you cannot isolate the cause of the trap, you should exit from ODT and start a new debugging session.

To help determine the cause of the trap, you can examine the task registers and stack before you start a new debugging session. Use the register name—the dollar sign ($) character followed by the register number—to access the task registers, as described in Section 5.1.7. To examine the stack, examine register 6 (the stack pointer) and use the at sign (@) character to open the location pointed to by the stack pointer. For example:

```
_$6/001200 @
001200 / 001216 RET
-
```

The slash character opens the stack pointer as a word and displays the address at the top of the stack. The at sign character takes the contents of the currently open location (that is, the stack pointer) as the address of the next location to be opened, opens it, and displays its contents, which is the top word on the stack.

To examine the stack, press the LINE FEED key to open and display each successive word on the stack. You can determine what the highest address of the stack can be by checking the line labeled Stack Limits in the task attributes section of the map. The line gives the following four numbers:

- Low address of the stack area
- High address of the stack area
- Octal extent of the stack area
- Decimal extent of the stack area

The high address tells you the last available location (that is, the bottom) of the stack. After you have examined the highest address, you have looked at all the items on the stack and can press the RETURN key to close the last available location.

To exit from the task using ODT, use the X (EXIT) command as follows:

```
_X
```

ODT performs the exit task directive and returns control to the Executive.

5.2 Using the Postmortem Dump

Another debugging aid is the Postmortem Dump (PMD). It requires no special code in your program. The example shows how to enable PMD for your task using the /POSTMORTEM qualifier. This qualifier sets a bit in the task flag word that causes a PMD whenever the task exits unexpectedly.

To enable PMD, type the following command line:

```
$ LINK/MAP/POSTMORTEM FILE,FILEA,FILEB RET
$
```

TKB is instructed to set a bit in the task flag word. This does not necessarily have to be done at the time you build the task. Postmortem dumps can also be specified as part of the RUN, INSTALL, and ABORT commands. (You can tell whether a task includes PMD by inspecting the task attributes section of the map. A line item called Task Attributes will have the designation PM.) When PMD is in effect for a task, the occurrence of an error that generates an SST causes the Executive to handle the termination of your task in a special manner. (This discussion assumes that the task does not handle SSTs through the SVTK$ directive and specially coded routines.) Instead of simply aborting the task, the Executive generates a request for PMD to create a formatted disk file showing the task image context. When a task generates an SST, the Executive initiates the normal task termination procedure (the printing of an error message and general register contents at the terminal) and, in addition, generates the request for PMD. To inform you that a dump is in effect, the Executive causes the following message to appear at the terminal:

```
Postmortem dump will be generated
```

PMD receives the request, creates a file in directory [1,4] on the library device, and generates a request to the spooler to print the file. The file has the name of the task and a type of PMD. The print spooler automatically deletes a file with the file type PMD after it is printed.

5.3 Using the Snapshot Dump

The snapshot dump ($SNAP) capability is a subset of PMD but requires special code in the task. Whereas PMD generates a dump of an entire task, the snapshot dump can produce a dump of only a portion of the task. Also, PMD generates a dump only when the task terminates abnormally, but the snapshot code can produce a dump at any place in the task execution. You include the necessary snapshot code in the task by editing the source file and inserting the snapshot macro calls where you want to produce a dump. (The snapshot macro calls the PMD task as described in the *RSX-11M-PLUS and Micro/RSX Task Builder Manual*). After you reassemble the modules containing the snapshot calls, you rebuild the task and substitute the reassembled modules. When you use snapshot macro calls, you do not need any special switches or options for TKB.

When you run the task and that section containing the special code is executed, a snapshot dump is taken. The special code generates a request for the PMD task. No special messages are printed at the terminal. To hold the dump, PMD creates a file with the name of the task and a file type of PMD in the directory that is the same as the User Identification Code (UIC) under which the task is running. PMD then generates a request for the spooling task to print and delete the file.

Chapter 6
Creating and Using Program Libraries

This chapter describes the procedures that create and maintain a library of macro source statements and a library of object module subroutines. It also shows how to include in your task image the macro call definitions and the object subroutines from user-created libraries.

The decision about whether to implement specific code as a macro call or as an object module subroutine is left to the designer. In general, the difference between implementations is in the tradeoff between assembly time and linking time and, secondarily, between convenience and size: Each time your source file invokes a specific macro call, the assembler must include the macro expansion in the object module. However, when your program calls an external subroutine, the resolution of the call is done during linking. Moreover, using the macro call to generate online code is convenient, but each invocation of the call increases the size of the resulting task image. However, if your program calls a specific external subroutine more than once, the subsequent invocations do not include that code in the task.

6.1 Creating and Using a Macro Source Library

The LIBRARY command creates and maintains library files that can contain macro definitions, object modules, or other elements. This section discusses creating a library file of macro definitions. Such a file has the default file type MLB and contains only macro definitions.

6.1.1 Creating the Macro Library

Use the LIBRARY/CREATE command to create a macro library from one input file of source definitions as follows:

```
$ LIBRARY/CREATE:(BLOCKS:25,MODULES:128)/MACRO [RET]
Library? USRMAC [RET]
Module(s)? USRMAC [RET]
$
```

or

```
$ LIBRARY/CREATE:(BLOCKS:25,MODULES:128)/MACRO USRMAC USRMAC [RET]
$
```

The /CREATE qualifier produces a library file named USRMAC.MLB. The file or files specified after the Module(s)? prompt are input to the library file. In Example 6–1, the input file is USRMAC.MAC.

Example 6–1: MACRO-11 Library Source Definitions

```
;
; SAVE - STORES REGISTER ON STACK
;
        .MACRO  SAVE,REG                ; PUSH REG ONTO STACK
        MOV     REG,-(SP)
        .ENDM
;
; RESTOR - POPS REGISTER VALUE OFF STACK
;
        .MACRO RESTOR,REG
        MOV      (SP)+,REG              ; POP REG OFF STACK
        .ENDM
        .END
```

The arguments to the /CREATE qualifier specify features of the library you are creating. Because there is more than one argument, each is enclosed in parentheses and separated by commas (,). The argument BLOCKS:25 gives the length in blocks for the library file. (DCL uses the decimal value automatically for all LIBRARY command arguments.) If you omit this argument, a file 100 blocks long is created by default. The argument MODULES:128 indicates the number of module name table (MNT) entries to allocate for this library. (Each macro definition in the library requires an entry in the MNT.)

The /MACRO qualifier identifies the type of library you wish to create. The default type is the /OBJECT qualifier (to create an object module library).

The Library? prompt requests that you name the library to be created. For macro libraries, the default file type is MLB. The Module(s)? prompt requests you to name the file or files containing the macro definitions. The default file type for this parameter is MAC. (If you do not name a file here, an empty file is created.)

When the macro library is created, the requested amount of contiguous file space is allocated. If sufficient contiguous space is not available, an "Open failure" error is generated and library creation terminates. To have the library created, you must either free up some space on the volume or try a smaller library size.

When the library file is created, an attempt is made to insert the macro definitions from the input file into the library. The input file is searched for .MACRO and .ENDM directives. If the macro definitions are nested, only the outermost directives are directly callable from the library. The name is extracted from each macro definition, and an entry is created in the MNT. The entry in the MNT is the means by which the assembler finds the associated macro definition in the library. Any code or comments outside the directives are discarded and all trailing blank and tab characters, blank lines, and comments are eliminated from the macro text itself. (This action, called squeezing, conserves memory for the assembler and reduces the space required to hold the macro definitions.) Errors occurring during the insertion of definitions usually indicate improper definitions, such as a missing .ENDM directive.

6.1.2 Using the Macro Definitions from the Library

Once the macro definitions are in the library, you need perform only three actions to have the assembler include the macro expansions in your code:

1. Include the name of the macro in a .MCALL directive in your program source file.
2. Invoke the macro call within the source file.
3. Specify the name of the library file in the command line to the assembler.

Thus, to invoke the two macro library definitions SAVE and RESTOR in your program, precede the macro calls themselves with a statement such as the following:

```
.MCALL   SAVE,RESTOR      ; CALL DEFINITIONS FROM USRMAC
```

This statement should occur at the start of the source file. When you assemble a source file that refers to a library file, you must name both files in your MACRO command line, as follows:

```
$ MACRO USRMAC/LIBRARY, USRTST/LIST [RET]
```

The name of the macro library can appear anywhere but last in the list of input files and must be marked with the /LIBRARY qualifier. The next file named is the first source file. To process the macro calls in the source file, the assembler uses the names given in the .MCALL directive to generate symbols for the macro symbol table. If you omit the name of the macro call from the .MCALL directive, the assembler cannot recognize the call itself in the code. (A corresponding entry is not in its macro symbol table.) It treats an unrecognized macro call as an implicit .WORD directive. If the macro name is not a valid symbol, its usage is flagged as an undefined reference by TKB. To expand the macro calls not defined in the source file, the assembler searches the library you specified before it searches the system default macro library. The MACRO-11 assembler does not search the system macro library for definitions that are found in the user library file.

6.2 Creating and Using an Object Module Library

The LIBRARY command may be used to create a library file containing object modules. Such a file has the file type OLB (object library) as a default and can contain only object modules.

6.2.1 Creating the Object Module Library

To create an object module library, you must have a file or files that contain the object modules to be inserted into the library. The following command lines create the object library and insert the modules FILEA.OBJ and FILEB.OBJ:

```
$ LIBRARY/CREATE:(BLOCKS:25,GLOBALS:128,MODULES:64)/OBJECT [RET]
Library? USROBJ [RET]
Module(s)? FILEA,FILEB [RET]
$
```

or

```
$ LIBR/CRE:(BLO:25,GLOB:128,MOD:64)/OBJ USROBJ FILEA,FILEB [RET]
$
```

The /OBJECT qualifier is not required because it is the default, but it is a good idea to include it. The default file type for an object module library is OLB. The default file type for the object module files is OBJ. The arguments to the /CREATE qualifier are the same as those used in creating a macro library, with the addition of the GLOBALS argument (which applies to object libraries only). The GLOBALS argument specifies the number of entry point table (EPT) slots to reserve. (An entry point is any global symbol in a module by which your program refers to the associated module.) If you do not supply a value, the default is GLOBALS:512. If you supply a value of 0, you can maintain modules with duplicate entry points in the same library. The names of the modules must still be unique. When building a library with GLOBALS:0, you must specify the correct module names to the Task Builder (TKB) when you build your task (see Section 6.2.2.). A good estimate for the number of EPT slots is twice the number of modules the library will contain. The value should be a multiple of 64. If not, the number is raised to the next multiple of 64. Again, all these numbers are decimal numbers in DCL.

When creating the object library file, the requested amount of contiguous space is allocated. You can estimate the number of contiguous blocks required by using the DIRECTORY command. Request a directory listing of all the files to be inserted in the library and use the total number of blocks displayed by the DIRECTORY command. If sufficient contiguous space is not available, an "Open failure" error is generated and library creation terminates. To have the library created, you must either free up some space on the volume or try to build a smaller object library.

When the object library is created, an attempt is made to insert, in the library, the object modules from the input file or files. The object library arranges the entries in the MNT in alphabetical order by module name. The module name used is the one you specified in the .TITLE directive when you assembled the object module. The module names and entry points must be unique. The global symbols in each object module are entered in the EPT. If a module name or an entry point is found that duplicates one already used, an error message is printed and processing stops.

If you suppress including entry points in the library EPT, you can insert, in the library, object modules having duplicate entry points. This feature enables you to maintain slightly different modules of the same general type in the same library. You select the correct module by specifying the unique module name to TKB when you build your task.

If an error is found, no modules are inserted in the library from the file containing the error. You must eliminate the error condition and insert the modules from the corrected file again. If no errors are found, all the modules are entered in the library. To ascertain what modules were inserted, obtain a listing of the library, as described in Section 6.3.3.

6.2.2 Using the Object Modules from the Library

When the object modules are in the library, you need perform only two actions to have TKB include the routines in your task.

1. Include the CALL x statement in the calling module (where x is an entry point to the called module). (It is assumed that the called module has a global statement to define the entry point.) CALL is a macro statement that is a permanent symbol in the MACRO-11 assembler. It standardizes subroutine calling conventions. CALL x translates to JSR PC,x (Jump to Subroutine program counter), where x is the subroutine entry point.
2. Specify the name of the library file and the names of the called modules in the command line to TKB.

Thus, to invoke subroutines from the library, ensure that the CALL statements are in your program. When you build a task, use a LINK command line similar to the following:

```
$ LINK/TASK:SUPLIB/MAP:SUPLIB [RET]
File(s)? FILE,USROBJ/INCLUDE:(TTREAD,TTWRIT) [RET]
$
```

or

```
$ LINK/TA:SUPLIB/MAP:SUPLIB FILE, USROBJ/INC:(TTREAD,TTWRIT) [RET]
$
```

By including file specifications as arguments to the /TASK and /MAP qualifiers, you cause the outfiles from the LINK command to be named SUPLIB.TSK and SUPLIB.MAP, respectively. The /INCLUDE qualifier identifies the file USROBJ.OBJ as an object library. The names appearing in parentheses, after the /INCLUDE qualifier, are the names of the modules to be extracted from the library and placed in the task. (Remember that these module names are derived from the names given in the .TITLE directive in the macro source files, and not from the file from which these modules were assembled.)

This method of specifying an object library search is more direct and faster than the method described in Section 6.2.3. If you are using a large library, TKB need only search the MNT for those object modules you specify. The disadvantage is that you have the responsibility to specify the names of all the modules that your task requires. If, however, you are using a library with zero entry points, the /INCLUDE qualifier is the only method of telling TKB which modules to include from that library.

6.2.3 Using the Library to Resolve Undefined Global Symbols

Often, the modules in a task refer to global symbols that are defined in other modules. If the modules that define the global symbols reside in a library, you can have TKB search the library. The /LIBRARY qualifier, specified with an input file specification for a LINK command, indicates that the entire library is to be searched. The /LIBRARY qualifier replaces the /INCLUDE qualifier as follows:

```
$ LINK/TASK:LB/MAP:LB FILE, USROBJ/LIBRARY [RET]
$
```

The /LIBRARY qualifier instructs TKB to search the library EPT for symbols that are referred to but not defined. When TKB finds a symbol in the table that is unresolved in the task, it extracts the defining module and places it in the task. If any symbols remain unresolved after the user library search, TKB searches the system library. This method requires less effort on your part than when you use the /INCLUDE qualifier. Note that if you are using a large library, the task build may take considerable time.

6.2.4 Dual Use of the Library

Under certain circumstances, you may want TKB to include specific modules from the library and also to search the same library to resolve any undefined references that may occur. For example, you may have conditional code in the main part of a task and not know what global symbols are referenced. TKB allows you to specify the two forms of the library search. This can be done by combining the /INCLUDE and /LIBRARY qualifiers in the same command line as follows:

```
$ LINK/TASK:LBOPT/MAP:LBOPT [RET]
File(s)? FILE, USROBJ/INCLUDE:TTREAD, USROBJ/LIBRARY [RET]
$
```

or

```
$ LINK/TASK:LBOPT/MAP:LBOPT FILE, USROBJ/INC:TTREAD, USROBJ/LIB [RET]
$
```

Once again, the arguments to the /TASK and /MAP qualifiers change the names of the associated output files. The /INCLUDE qualifier on the file specification for USROBJ.OLB instructs TKB to extract the named module. Notice that because only one module is named, the parentheses are not necessary. The /LIBRARY qualifier in the file specification for USROBJ.OLB instructs TKB to search that library for any unresolved global symbols. TKB includes in the task any modules from the library that are unresolved at that point in the task build. If any unresolved symbols remain after the search of the user library, TKB searches the system library.

6.3 Maintaining User Libraries

This section decribes the following three simple operations used to maintain a user library:

- Adding modules to a library
- Replacing a module in a library
- Obtaining information about a library

This is done using the LIBRARY/INSERT, LIBRARY/REPLACE, and LIBRARY/LIST commands.

6.3.1 Adding Modules to a Library

Modules can be added to a library with the LIBRARY/INSERT command as follows:

```
$ LIBRARY/INSERT [RET]
Library? USRMAC.MLB [RET]
Module(s)? MAC1,MAC2 [RET]
$
```

or

```
$ LIBRARY/INSERT USRMAC.MLB MAC1,MAC2 [RET]
$
```

Type the name and type of the library in response to the Library? prompt. At the Module(s)? prompt, type in the names of the files containing the library modules. The default file type for files containing library modules is the same as the library type you specified.

You cannot add modules to a library that has no remaining entries in the MNT. (If you are creating an object module library, there must be sufficient EPT slots as well.) When a module is inserted in a library, a check is made to be sure that a module of the same name does not currently reside in the library. If such a module is found, an error is reported and nothing is done. (For inserting object modules, a check is also made for duplicate entry point names.) To add modules with duplication, see the discussion of LIBRARY/REPLACE command in Section 6.3.2.

6.3.2 Replacing a Module in a Library

After you create a library, you will need to change and update modules in the library from time to time. Because a module of the same name (and, for object modules, the same entry points) already exists, you must perform a replace operation. This is done with the LIBRARY/REPLACE command as follows:

```
$ LIBRARY/REPLACE [RET]
Library? USROBJ [RET]
Module(s)? FILEA [RET]
Module "TTREAD" replaced

$
```

or

```
$ LIBRARY/REPLACE USROBJ FILEA [RET]
Module "TTREAD" replaced

$
```

This command line logically deletes the module TTREAD and all associated entry points for that name from USROBJ.OLB. The new version of module TTREAD is inserted from FILEA.OBJ and a message is printed. If a module to be replaced is not found in the library, an insertion is performed but no message is printed.

Note that LIBRARY/REPLACE command causes a logical deletion and does not reclaim the space occupied by the module you replace. To reclaim this lost space, you should occasionally use the LIBRARY/COMPRESS command. (See the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2* for information about the LIBRARY command.)

6.3.3 Obtaining Information About a Library

To obtain information about a library, type a LIBRARY/LIST command in the following format:

```
$ LIBRARY/LIST:LBLIST/NAMES/FULL [RET]
Library? DD.OLB [RET]
$
```

or

```
$ LIB/LIS:LBLIST/NAMES/FULL DD.OLB [RET]
$
```

This command line accesses the library file DD.OLB. The list appears in the file in your directory called LBLIST.LST. The /FULL qualifier lists entry points and full information (size, date of creation, and, for object modules, identification).

To list the information on the terminal instead of a file, use the LIBRARY/LIST command without a file specification argument to the /LIST qualifier as follows:

```
$ LIBRARY/LIST/FULL USRMAC.MLB [RET]

(Information listed)
$
```

Chapter 7
Using Logical Name Commands

7.1 Logical Names

This chapter provides information on using logical name commands. The syntax of the following commands is given in detail:

- ASSIGN
- DEASSIGN
- DEFINE
- SHOW ASSIGNMENTS and SHOW LOGICALS

A logical name is a user- or system-defined name for an equivalence name, which can be the following:

- All or part of a file specification—This keeps your programs and command procedures independent of physical file specifications.
- A physical device—You can assign logical names to devices such as magnetic tape drives, terminals, and line printers. The system manager can assign logical names to public disk volumes so that users do not have to be concerned with the physical location of these volumes.
- Anything created by the DEFINE command (see Section 7.4 for more information).

To reduce typing, you can use logical names as a shorthand way of specifying files or directories that you refer to frequently. For example, you might assign the logical name HOME to your default disk and directory, or the logical name DIARY to a file in which you keep a log of your daily activities. You can also use logical names in file specifications to keep your programs and command procedures independent of physical file specifications.

7.1.1 Logical Name Tables

The system maintains logical name and equivalence name pairs in the following four logical name tables:

Task logical name table
Contains logical name entries that are created for an individual task using the Create Logical (CLOG$/CLON$) directive. These entries remain in the table for only as long as the task is running. When either the task has completed execution or the Delete Logical (DLOG$/DLON$) directive has been issued, the logical names are removed from the table. (See the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* for more information on the CLOG$/CLON$ and the DLOG$/DLON$ directives.)

User logical name table
Contains logical name entries that are local to a particular user or terminal session. By default, the DEFINE and ASSIGN commands and login assignments place a logical name in the session logical name table.

Group logical name table
Contains logical name entries that are qualified by a group number. These entries can be accessed only by tasks that execute with the same group number in their User Identification Codes (UICs) as the user that assigned the logical name. You must use the /GROUP qualifier to make an entry in the group logical name table.

System logical name table
Contains entries that can be accessed by any task in the system. You must use the /SYSTEM or /GLOBAL qualifier to make an entry in the system logical name table.

You must be privileged to place entries in the group and system logical name tables.

7.1.2 Displaying Logical Name Table Entries

The SHOW LOGICALS and SHOW ASSIGNMENTS commands display current entries in the logical name tables.

To display the contents of the session logical name table, enter the SHOW LOGICALS or SHOW ASSIGNMENTS command, without any qualifiers or parameters as follows:

```
$ SHOW LOGICALS [RET]
```

or

```
$ SHOW ASSIGNMENTS [RET]
```

These commands produce a display of the current logical names in the session logical name table and their equivalence.

You can request the system to display all entries in the specific logical name table. For example:

```
$ SHOW LOGICALS/SYSTEM [RET]
$ SHOW LOGICALS/GLOBAL [RET]
$ SHOW LOGICALS/GROUP:g [RET]
$ SHOW LOGICALS/LOCAL [RET]
$ SHOW LOGICALS/LOGIN [RET]
$ SHOW LOGICALS/ALL [RET]
$ SHOW LOGICALS/TERMINAL:ttnn: [RET]
```

The SHOW LOGICALS/SYSTEM and the SHOW LOGICALS/GLOBAL commands display all the global logical assignments in the system.

The SHOW LOGICALS/GROUP command displays the logical names accessible to users with the same group number. The argument g allows you to specify a specific group number. (This is a privileged command.)

The SHOW LOGICALS/LOCAL command displays the local and login logical assignments.

The SHOW LOGICALS/LOGIN command displays only the login logical assignments.

The SHOW LOGICALS/ALL command displays the system, your group, local, and login logical assignments.

The SHOW LOGICALS/TERMINAL:ttnn: command displays local and login logical assignments for the specified terminal (ttnn). The /TERMINAL qualifier may be used in conjunction with the /LOCAL, /LOGIN, and /ALL qualifiers.

7.1.3 How to Create and Delete Logical Names

Logical names and equivalence name strings can each have a maximum of 255 characters, and they can be used to form all or part of a file specification. If only part of a file specification is a logical name, it must be the leftmost component of the file specification. You can then specify the logical name in place of the device (or device and directory name) in subsequent file specifications, terminated by a colon (:).

When you specify an equivalence name for the ASSIGN command, you must specify it using the proper punctuation marks (colons, brackets, periods). If you specify only a device name, you must terminate the equivalence name parameter with a colon; if you specify a device and directory name, or a full file specification, do not terminate the equivalence name with a colon.

You can optionally terminate a logical name with a colon. If you do this, the ASSIGN command removes the colon before placing the logical name in the logical name table. The DEFINE command, on the other hand, does not remove the colon before placing the logical name in the logical name table. However, if you specify a colon at the end of the logical name, and if you want the colon as part of the logical name, use the DEFINE command to create the logical name.

To delete a logical name, use the DEASSIGN command. Generally, to delete a logical name created by the DEFINE command, you should put quotation marks (") on both sides of the logical name in the DEASSIGN command line. For example, to delete the logical name TASK (created by the DEFINE command), use the following command line:

```
$ DEASSIGN "TASK:" [RET]
```

You do not need the quotation marks to delete any logical names created by the ASSIGN command.

7.1.4 Logical Name Translation

For logical names created using the ASSIGN command, if the system finds a logical name, it substitutes the equivalence name for the logical name in the file specification. This is called logical name translation.

When the system translates logical names, it searches the task, user, group, and system tables (in that order), and uses the first match it finds.

If you are not sure about the equivalence name assigned to a logical name, use either the SHOW LOGICALS or SHOW ASSIGNMENTS command.

7.1.5 Iterative Translation

When the system translates logical names in file specifications, the logical name translation can be iterative. This means that after the system translates a logical name in a file specification, it repeats the process of translating the file specification. For example, consider the following logical name table entries made with the ASSIGN commands:

```
$ ASSIGN LB:[3,54] RANDOM [RET]
$ ASSIGN RANDOM:VMR.TSK TASK [RET]
```

The first ASSIGN command equates the logical name RANDOM to the device and directory specification LB:[3,54]. The second ASSIGN command equates the logical name TASK to the equivalence string RANDOM:VMR.TSK. In subsequent commands, or in programs you execute, you can refer to the logical name TASK. For example:

```
$ RUN TASK [RET]
```

When the system translates the logical name TASK, it finds the equivalence string RANDOM:VMR.TSK. The system then checks to see if the portion to the left of the colon (if there is a colon) in the equivalence name is a logical name. If it is a logical name (as RANDOM is in this example), the system translates that logical name also. When the logical name translation is complete, the translated device and file specification is LB:[3,54]VMR.TSK.

Note

The system limits logical name translation to 10 levels. If you define more than 10 levels or create a circular definition, an error occurs when the logical name is used.

You can control the translation of logical names by using the /FINAL qualifier on either the ASSIGN or DEFINE command. For example, you assign three levels of logical names in the following manner:

```
$ ASSIGN DB0:[TSTSYS] TEST5: [RET]
$ ASSIGN/FINAL TEST5: LAST: [RET]
$ ASSIGN LAST: DISK: [RET]
```

Without the /FINAL qualifier on the second assignment, the logical name DISK would eventually be translated into the equivalence name DB0:[TSTSYS]. Because you applied the /FINAL qualifier to the second assignment, the translation stops at the equivalence name TEST5. The task is less hardware dependent when you use this qualifier. If for some reason, DB0 is no longer available

for use, you can assign TEST5 to point to another disk drive (when the task is not running) without having to change the task itself. However, at some point, you (or the task) have to refer to the logical name TEST5 (for example, spawning another task, or issuing a ACHN$ directive) in order for the translation from DISK to DB0:[TSTSYS] to work. For more information on the ACHN$ directive, see the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual.*

7.2 ASSIGN Command

The ASSIGN command equates a logical name to a Files-11 physical device name, to all or part of a Files-11 file specification, or to another logical name. All references to the logical name are resolved by the operating system.

Formats

```
$ ASSIGN
Logical name equivalent?  equivalence_name
Logical name?  logical_name
```

or

```
$ ASSIGN[/qualifier[s]] equivalence_name logical_name
```

Command Qualifiers

/FINAL
/GLOBAL
/GROUP[:g]
/LOCAL
/LOGIN
/SYSTEM
/TERMINAL:ttnn:
/TRANSLATION:FINAL

Parameters

equivalence_name

Specifies the Files-11 device or file specification that you have defined as the substitution for the logical name. The equivalence name can also be another logical name that will be iteratively invoked.

The ASSIGN command checks the syntax of an equivalence name that is a device or file specification. When you specify an equivalence name that will be used as a file specification, you must include the punctuation marks (colons, brackets, periods) that would be required if the equivalence name were used directly as a file specification. Therefore, if you specify a device name as an equivalence name, terminate the device name with a colon. The ASSIGN command will not remove the terminating colon. If you did not correctly specify the device or file, the ASSIGN command fails. Also, if you specify quotation marks (") around the equivalence name, the ASSIGN command retains the quotation marks. (This differs from the DEFINE command, which removes the quotation marks.)

logical_name

Specifies the name you selected that you want to give the device or file specification.

If you terminate the logical name with one colon (:) or two colons (::), the system removes the colons before placing the name in the logical name table. (This differs from the DEFINE command, which retains the colons.) The ASSIGN command removes the colons because FCS-11 and RMS-11 file access methods do not consider the terminating colons to be part of the logical name that either file access method attempts to process.

A logical name may contain the following ASCII characters: the 26 letters A to Z; the numbers 0 to 9; and the characters underscore (_), colon (:), and dollar sign ($).

The length of the logical name is limited by the legal length of a command line, which is 255 characters. For example, if the length of the ASSIGN command plus qualifiers plus the equivalence name plus spaces is 24 characters, the logical name cannot be more than 231 characters long.

The ASSIGN command will do complete logical translation of the equivalence string and create the logical with the /FINAL qualifier. The logical name and equivalence name consist of a device in ddnn format. The physical device specified in the equivalence name must exist in the system. These guidelines provide compatibility with previous logical name capability.

The ASSIGN command will also do zero compression on the logical and equivalence names.

Command Qualifiers

/GLOBAL
/SYSTEM

Specifies that the assignment is to be a system table assignment. The /SYSTEM and /GLOBAL qualifiers are synonyms and are privileged qualifiers. System assignments apply to all tasks running on the system.

/GROUP[:g]

Specifies that the assignment is to be a group assignment.

The argument g is the UIC group number of the users who share the logical name. If you do not specify a group number, the default is your own group number, which is taken from your current protection UIC. If you are nonprivileged, the group number is same as the UIC that is assigned to you when you log in.

/LOCAL

Specifies that the assignment is to be a local assignment. This is the default qualifier, so you do not need to specify it.

Commands and tasks initiated from your terminal can access devices or files through the logical names assigned to them. Note that no automatic deassignment occurs if you dismount a device after assigning a logical name to it.

You can define your own set of local assignments for your terminal. These local logical names exist only for your terminal.

/LOGIN

Specifies that the assignment is to be a login assignment. This is a privileged qualifier.

Login assignments are usually established through ACNT, the Account File Maintenance Program. However, the ASSIGN/LOGIN command does not alter the account file. These logical names exist for your account, and they are available to you regardless of the terminal on which you log in.

When a user issues a LOGIN command to log in to the system, the system automatically assigns the logical name SYS$LOGIN to the user's default device and directory, which is the device and directory that contains the user's files.

/TERMINAL:ttnn:

Specifies that the requested local assignment be applied to another terminal. Only a privileged user may make assignments to other terminals. Note that the target terminal must be logged in before the assignment can be made.

/FINAL
/TRANSLATION:FINAL

Specifies that the equivalence name string should not be translated iteratively; that is, the logical name translation should terminate with the current equivalence string. See Section 7.1.4 for more information on the iterative translation of logical names. The /TRANSLATION:FINAL qualifier is a synonym for the /FINAL qualifier, which is included for VMS compatibility.

Examples

```
$ ASSIGN RET
Logical name equivalent? DU1: RET
Logical name? TP1: RET
```

Assigns the logical name TP1 to the physical device DU1. The user may now issue commands referring to device TP1 (in any command that accepts a device specification) and DU1 will be substituted for it.

```
$ ASSIGN LB0: RR2: RET
```

Assigns the logical name RR2 to the pseudo device LB0. This logical name exists for your terminal only.

```
$ ASSIGN/GROUP:303 DU0:[1,1]SYSLIB.OLB;3 SYS$LIB RET
```

Assigns the logical name SYS$LIB to file SYSLIB.OLB;3, which is located on device DU0 in directory [1,1]. This logical name exists for users whose UIC group number is 303.

```
$ ASSIGN/TERMINAL:TT4: DU2:[TEST] A: RET
$ SHOW ASSIGNMENTS/TERMINAL:TT4: RET
A = DU2:[TEST] (Local, TT4:)
```

Assigns the logical name A to the directory [TEST], which is located on device DU2 for all commands and tasks initiated from TT4. The user then issues a SHOW ASSIGNMENTS command to display the logical name. These commands must be issued from a privileged terminal. Also, the terminal TT4 must be logged in before the assignment can be made.

```
$ ASSIGN/GLOBAL DL1: XX1: RET
```

Assigns the logical name XX1 to the physical device DL1. All users and tasks on the system can refer to XX1 when they initiate commands and tasks. This command must be issued from a privileged terminal. The string DL1 can also be used to reference the physical device DL1.

```
$ ASSIGN DU1 DQ RET
$ SHOW ASSIGNMENTS RET
DQ = DU1: (Local, TT:)
```

Assigns the logical name DQ to the physical device DU1. Although the user did not terminate the equivalence name (DU1) with a colon, DCL recognizes that the equivalence name resembles the former ASSIGN command device format (ddnn). Because the ASSIGN command line is in this format, DCL automatically terminates the equivalence name string with a colon, as shown when the user issued the SHOW ASSIGNMENTS command.

```
$ ASSIGN DU0:[TSTSYS] TEST5: RET
$ ASSIGN/FINAL TEST5: LAST: RET
$ ASSIGN LAST FIRST RET
```

Shows three assignments. First, the user assigns logical name TEST5: to the equivalent name DU:[TSTSYS]. Second, the user then assigns the logical name LAST to the equivalent name TEST5: with the /FINAL qualifier. Third, the user assigns the logical name FIRST to the equivalent name LAST. When a task (for example) refers to the logical name FIRST, the translation is carried only as far as the equivalent name TEST5: and then stops.

Notes

1. The order of precedence in logical names, from highest to lowest, is: task, local, login, group, and global. This means that if the logical name DEV has a global assignment of DU1 but a local assignment of DU2, the operating system interprets DEV to be DU2 for your terminal.
2. The ASSIGN command is counteracted by the DEASSIGN command.
3. You can display current assignments with either the SHOW ASSIGNMENTS or SHOW LOGICALS command.
4. The logical name SY0 can be assigned only to a device.
5. The ASSIGN/REDIRECT command is described in the *Micro/RSX User's Guide, Volume 1* and the *Micro/RSX User's Guide, Volume 2*.
6. The ASSIGN/TASK command is described in the *Micro/RSX User's Guide, Volume 1* and the *Micro/RSX User's Guide, Volume 2*.
7. The ASSIGN/QUEUE command is described in the *RSX-11M-PLUS and Micro/RSX System Management Guide* and the *Micro/RSX User's Guide, Volume 1*.

Error Messages

ASS—Device not in system

Explanation: The specified equivalence device name does not exist in the system.

User Action: Specify a device that is recognized by the system and reenter the command line.

ASS—Device not terminal

Explanation: You did not specify a terminal when you issued the ASSIGN/TERMINAL command.

User Action: Specify a terminal and reenter the command line.

ASS—Terminal not logged in

Explanation: The terminal you attempted to make a logical assignment to is not logged in.

User Action: Log in the terminal you want to make the assignment for and reenter the command line.

ASS—Octal group number expected

Explanation: You did not specify an octal group number value when you issued the DEFINE/GROUP command.

User Action: Check the group number that you want and reenter the command.

7.3 DEASSIGN Command

DEASSIGN cancels logical name assignments made by the ASSIGN and DEFINE commands.

Format

$ DEASSIGN[/qualifier[s]] logical_name

Command Qualifiers

/ALL
/GLOBAL
/SYSTEM
/GROUP[:g]
/LOCAL
/LOGIN
/TERMINAL:ttnn:

Parameter

logical_name

Specifies the logical name assignment that you want to delete. This parameter is required with all qualifiers except the /ALL qualifier.

If you terminate the logical name parameter with one colon (:) or two colons (::), the command interpreter removes the colons. (Note that the ASSIGN command, also, removes trailing colons, if present, from a logical name before placing the name in the logical name table.) If a colon is present at the end of the actual logical name, you must place quotation marks (") around the logical name parameter for the DEASSIGN command (for example, DEASSIGN "file:").

A logical name can contain any ASCII characters. However, if a logical name includes characters other than the 26 letters A to Z; the numbers 0 to 9; and the characters underscore (_), colon (:), or the dollar sign ($); you must place a quotation mark on each side of the logical name. For example, the logical name INFILE does not require quotation marks, but you must specify the logical name C3_PO in the command line as "C3_PO". Generally, any logical name created by the DEFINE command should have the quotation marks placed on either side of the logical name in the DEASSIGN command line.

For systems *without* extended logical name support, logical_name specifies the logical device name. This is a required parameter except with the /ALL qualifier. Logical device names have the same format as all other device names, a two-letter mnemonic followed by an octal number terminated by a colon.

Command Qualifiers

Any other qualifier can be used with the /ALL qualifier. The /TERMINAL qualifier can be used with the /LOGIN or /LOCAL qualifier.

/ALL

Deletes all logical name assignments for a particular table.

If you use the /ALL qualifier with another qualifier, all assignments of the type specified by the other qualifier are deleted.

Because the /ALL qualifier deletes all the logical name assignments in a given category, you do not specify the logical name parameter in the command line.

/GLOBAL
/SYSTEM

Deletes a global logical name assignment in the system logical name table. This deassignment applies to all tasks running in the system. The /SYSTEM and /GLOBAL qualifiers are synonyms and are privileged qualifiers. You cannot specify the /TERMINAL qualifier with the /GLOBAL qualifier.

/GROUP[:g]

Deletes a group logical name assignment in the group logical name table. The argument g identifies the User Identification Code (UIC) group number for which the logical name exists. If you do not specify a UIC group number, the default is your own group number, which is taken from your current protection UIC. If you are nonprivileged, the group number is same as the UIC that is assigned to you when you log in.

/LOCAL

Deletes a local logical name assignment in the group logical name table. The /LOCAL qualifier is the default.

/LOGIN

Deletes a login logical name assignment. The /LOGIN qualifier is privileged.

/TERMINAL:ttnn:

Allows you to delete a logical name assignment of another terminal. You cannot specify the /TERMINAL qualifier with the /GLOBAL qualifier. Only privileged terminals can delete assignments made from other terminals.

Examples

```
$ DEASSIGN TP0: RET
```

Deletes the local assignment of logical name TP to a device. The DEASSIGN command ignores the terminating colon.

```
$ DEASSIGN/ALL RET
```

Deletes all local logical name assignments.

```
$ DEAS/LOCAL/ALL RET
```

Provides the same results as the previous example. The /LOCAL qualifier is the default.

```
$ DEAS/LOCAL/ALL/TERMINAL:TT4: RET
```

Deletes all local assignments for terminal TT4. This command must be issued from a privileged terminal.

```
$ DEASSIGN "XY:" RET
```

Deletes the logical name XY (created with the DEFINE command).

Notes

1. The DEASSIGN command counteracts ASSIGN and DEFINE commands.
2. You must type at least the first four characters of the DEASSIGN command.
3. Login assignments are normally established through the Account File Maintenance Program (ACNT).
4. You can display assignments with either the SHOW ASSIGNMENTS or SHOW LOGICALS command.
5. All local assignments are deassigned when you either log out or log in.
6. The DISMOUNT command does not delete logical name assignments.
7. The DEASSIGN/QUEUE command is described in the *Micro/RSX User's Guide, Volume 1.*

Error Messages

DEA—Device not terminal

Explanation: The /TERMINAL qualifier named a device that is not a terminal.

User Action: Retype command after checking for proper syntax.

DEA—Octal group number expected

Explanation: You did not specify an octal group number value when you issued the DEFINE/GROUP command.

User Action: Check the group number that you want and reenter the command.

7.4 DEFINE Command

The DEFINE command equates a logical name to an explicit ASCII text string (for example, "$_TESTFILE @@6") or to another logical name. All references to the logical name are resolved by the operating system.

Note that the DEFINE command does not perform validity checks for node names, device names, User File Directory (UFD) specifications, or file specifications. If you intend to use a logical name as part of a Files-11 file specification, you should use the ASSIGN command to create the logical name. The ASSIGN command performs several validity checks to ensure that the logical name will be recognized by the system.

Formats

```
$ DEFINE
Logical name?    logical_name
Equivalent name string?    equivalence_name
```

or

```
DEFINE[/qualifier[s]] logical_name equivalence_name
```

Command Qualifiers

/GLOBAL
/SYSTEM
/GROUP[:g]
/LOGIN
/LOCAL
/TERMINAL:ttnn:
/FINAL
/TRANSLATION:FINAL

Parameters

logical_name

Specifies the selected name to give the device or file specification.

A logical name can contain any ASCII characters. However, if a logical name includes characters other than the 26 letters A to Z; the numbers 0 to 9; or the characters underscore (_), or colon (:), or the dollar sign ($); you must place quotation marks (") on each side of the logical name. For example, although the logical name INFILE does not require quotation marks, you must specify the logical name C4PO* in the command line as "C4PO*". Unlike the ASSIGN command, the leading and trailing quotation marks are stripped off when the logical assignment occurs.

If you specify a colon at the end of the logical name, the DEFINE command saves the colon as part of the logical name. (This is in contrast to the ASSIGN command, which removes the colon before placing the name in the logical name table.) Note, however, that the system will ignore all logical names terminating with a colon when processing file specifications.

The length of the logical name is limited by the legal length of a command line, which is 255 characters. For example, if the length of the DEFINE command plus qualifiers plus the equivalence name plus spaces is 24 characters, the logical name cannot be more than 231 characters long.

equivalence_name

Specifies an ASCII string (for example, any user-specified text) that you have defined as the substitution for the logical name. The equivalence name can also reference another logical name or a text string.

Command Qualifiers

/GLOBAL
/SYSTEM

Specifies the assignment is to be a system table assignment. The /SYSTEM and /GLOBAL qualifiers are synonyms and are privileged commands. System assignments apply to all tasks running in the system.

/GROUP[:g]

Specifies that the assignment is to a group assignment.

The argument g is the User Identification Code (UIC) group number of the users who share the logical name. If you do not specify a group number, the default is your own group number, which is taken from your current protection UIC. If you are nonprivileged, the group number is same as the UIC that is assigned to you when you log in.

/LOCAL

Specifies that the assignment is to be a login assignment. This is the default qualifier, so you do not need to specify it.

Commands and tasks initiated from your terminal can access devices and files through the logical names defined for them. Note that no automatic deassignment occurs if you dismount a device after defining a logical name for it.

You can define your set of logical name assignments for your own terminal. Those local logical names exist only for your terminal.

/LOGIN

Specifies that the logical definition is to be displayed as a login definition. This is a privileged qualifier.

You can establish login definitions any time during an individual user session. These definitions remain in effect until you log out of the system or you specifically delete the definition by using the DEASSIGN/LOGIN command.

Normally, you place login definitions in your login command file, LOGIN.CMD, or the system manager places them in the system login file, SYSLOGIN.CMD. Having the logical definitions in either of these login command files saves you having to define those logicals each time you log in. For more information on the SYSLOGIN.CMD file, see the *Micro/RSX System Manager's Guide.*

/TERMINAL:ttnn:

Specifies that the requested local assignment be applied to another terminal. Only a privileged user may make assignments to other terminals. Note that the target terminal must be logged in before the assignment can be made.

/FINAL
/TRANSLATION:FINAL

Specifies that the equivalence name string should not be translated iteratively; that is, the logical name translation should terminate with the current equivalence string. See Section 7.1.4 for more information on the iterative translation of logical names. The /TRANSLATION:FINAL qualifier is a synonym for the /FINAL qualifier, which is included for VMS compatibility.

Examples

```
$ DEFINE [RET]
Logical name? "TOM'S TEST" [RET]
Equivalent name string? "BLEW UP" [RET]
$ SHOW LOGICALS [RET]
TOM'S TEST = BLEW UP (local, TT0:)
```

Defines the logical name TOM'S TEST to the equivalent string BLEW UP. The user issued a SHOW LOGICALS command to verify that the assignment had been made. The DEFINE command removes the quotation marks from the logical name.

```
$ DEFINE "TOM'S TEST"  "BLEW UP" [RET]
```

Provides the same results as the previous example.

Notes

1. The order of precedence in logical names, from highest to lowest, is: task, local, login, group, and global. This means that, if the logical name BIG RIVER @ BEND has a global assignment of RUNS SLOWLY, but a local assignment of RUNS QUICKLY, the operating system interprets BIG RIVER @ BEND to be RUNS QUICKLY for your terminal.
2. You can counteract the DEFINE command by using the DEASSIGN command or by redefining the logical name.
3. You can display current assignments with either the SHOW ASSIGNMENTS or the SHOW LOGICALS command.

Error Messages

DEF—Device not terminal

Explanation: You did not specify a terminal when you issued the ASSIGN/TERMINAL command.

User Action: Specify a terminal and reenter the command line.

DEF—Function requires logical name support

Explanation: Your system does not support extended logical name support.

User Action: None.

DEF—Octal group number expected

Explanation: You did not specify an octal group number value when you issued the DEFINE/GROUP command.

User Action: Check the group number that you want and reenter the command.

DEF—Terminal not logged in

Explanation: The terminal you attempted to make a logical assignment to is not logged in.

User Action: Log in the terminal you want to make the assignment for and reenter the command line.

7.5 SHOW ASSIGNMENTS and SHOW LOGICALS Commands

The SHOW ASSIGNMENTS and SHOW LOGICALS commands display, at your terminal, all local and login logical assignments. Privileged users can display assignments for other terminals as well as all assignments in the operating system.

Logical assignments are established by the ASSIGN, DEFINE, and SET DEFAULT commands, and by ACNT. See the *RSX-11M-PLUS and Micro/RSX System Management Guide* for more information on ACNT.

Formats

SHOW ASSIGNMENTS[/qualifier]

or

SHOW LOGICALS[/qualifier]

Command Qualifiers

/ALL
/GLOBAL
/SYSTEM
/GROUP[:g]
/LOCAL
/LOGIN
/TERMINAL:ttnn:

Command Qualifiers

/ALL

Displays all of your local, login, and group logical name assignments, as well as all global assignments.

You can also use this qualifier with the /TERMINAL:ttnn: qualifier to see the local, login, and group logical names for terminal ttnn displayed on your terminal.

/GLOBAL
/SYSTEM

Specifies that all global logical assignments in the operating system are to be displayed on your terminal. The /SYSTEM and /GLOBAL qualifiers are synonyms and are privileged commands. System assignments apply to all tasks running in the system.

/GROUP[:g]

Displays the group logical assignments for users with the specified User Identification Code (UIC) group number, g.

Nonprivileged users can see the group logical assignments of their own group. In this case, you can either specify the /GROUP qualifier without an argument or use your own group number as the argument.

Only privileged users can see the logical assignments of other groups.

/LOCAL

Specifies that login logical assignments for your terminal are to be displayed on your terminal. This qualifier is the default.

/LOGIN

Specifies that local and login logical assignments for your terminal are to be displayed on your terminal. The /LOGIN qualifier is privileged.

/TERMINAL:ttnn:

Specifies that local and login logical assignments for terminal ttnn are to be displayed on your terminal. This is a privileged qualifier.

You can also use this qualifier with the /ALL qualifier to see all of the logical assignments for terminal ttnn.

Examples

```
$ SHOW [RET]
Function? ASSIGNMENTS [RET]
LP = DU1: (Local, TT:)
INFILE = DUO:[DANIEL]ADDRESS.TXT (Local, TT:)
R3D2 = A Robot (Login, TT:)
SYS$LOGIN = DU:[SAMUEL] (Login, Final, TT:)
```

Displays your local and login logical assignments. The logical name is displayed first, followed by the equivalence name. Finally, the type of assignment and your terminal number are given.

You can use these logical names in place of device and file specifications. References to LP will actually go to DU1. The file ADDRESS.TXT, which is located on device DU in directory [DANIEL], is an input file for a task that specifies the logical name INFILE. In addition, the string "A Robot" is the translation of the logical name R3D2. Finally, the user has the login assignment of SYS$LOGIN to the device DU and directory [SAMUEL].

```
$ SHOW ASSIGNMENTS/LOCAL [RET]
```

Provides the same results as the previous example.

```
$ SHOW ASSIGNMENTS/ALL [RET]
WK = LB: (Global, Final)
TEXT = DU:[SYSTST]LOGIN.TXT (Local, TT:)
SYS$LOGIN = DU:[QUERY] (Login, Final, TT:)
```

Displays all the logical assignments (local, login, global, and system) for your terminal. The system has given the global logical name WK to the pseudo device, LB, and the login logical name SYS$LOGIN to the directory, DU:[QUERY]. The user has given the local logical name TEXT to the file specification, DU:[SYSTST]LOGIN.TXT.

```
$ SHOW ASSIGNMENTS/TERMINAL:TT3: [RET]
MP = SY: (Local, TT3:)
```

Displays the logical assignments for terminal TT3 and requires a privileged terminal and a terminal (TT3) that is logged in. This user has given the local logical name MP to the pseudo device, SY.

```
$ SHOW ASSIGNMENTS/GLOBAL [RET]
WK = LB: (Global, Final)
IN = SY: (Global, Final)
EX = SY: (Global, Final)
```

Displays all global logical names and requires a privileged terminal.

Chapter 8

Guidelines for Creating an Optional Software Package

This chapter explains how to create an optional software diskette or tape package that a user can install on a Micro/RSX operating system. The first step in producing your package is to create your optional software kit on diskette. (To create an optional software kit on tape, you must still create the diskette kit first). A complete optional software package includes an installation guide for the user.

The information in this chapter is intended for programmers who are developing and/or packaging optional software for Micro/RSX systems and who are familiar with Micro/RSX concepts. Programmers creating optional software kits should also be experienced in the following areas:

- Developing Micro/RSX software applications
- Using EDT
- Using the following DCL Commands:
  - ANALYZE/MEDIA
  - INITIALIZE
  - COPY
  - BACKUP

In this chapter, these terms are used as follows:

Optional software
Refers to one or more of the following:

- An optional component of Micro/RSX
- A DIGITAL-supplied layered product
- A third-party software application

This chapter uses FORTRAN-77 examples to create the optional software kit. However, by using the guidelines presented in this chapter and the prompts requesting option or file names, you can create any option.

Caution

The FORTRAN-77 examples in this chapter are for illustration purposes only and *do not* represent the DIGITAL product Micro/RSX FORTRAN-77.

Kit
Specifies a set of diskettes or tapes, which contains one or more options to be installed.

User
Specifies the person installing the diskette or tape kit at the installation site.

Install
Specifies the procedure that physically includes on the fixed disk all the files necessary to support an option, run task images, execute commands, or invoke an indirect command file. The install procedure is invoked by the user.

Remove
Specifies the procedure that disables a previously installed option. These steps include deleting a set of files copied to the fixed disk during installation, deleting unnecessary files created by the option, deleting task images from the system, or invoking an indirect command file. The remove procedure is invoked by the user.

Package
Specifies the complete software option that includes the diskette or tape kit and installation guide.

8.1 Preparing to Create a Diskette Package

This section describes the resources you need and the steps you must follow to create an optional software diskette kit (which combined with your documentation completes the diskette package). The resources you need are as follows:

- A Micro/RSX Version 4.0 operating system
- A copy of the optional software files
- Blank RX50 diskettes

Once you have the necessary resources, you must perform the following steps to create the optional software diskette package:

1. Determine how many diskettes you need.
2. Determine how you will group the optional software files to form your diskette kit.
3. Set up the INSTALL.DAT file and the option INS file.
4. Create the INSTALL.DAT file.
5. Create the option INS file.

6. Create the diskette kit.
7. Document the installation procedure.

The following sections describe each step in detail, beginning with Section 8.1.1.

8.1.1 Determining Required Number of Diskettes

Before you create your kit, determine how many diskettes you need by listing the files to be included on the optional software kit and the directories in which they reside. You will need one diskette for every 770 blocks of information.

8.1.2 Grouping Optional Software Files

If you need more than one diskette, you should determine how you will group your files before you create the kit. Grouping the files by function makes it easier to create the kit. For example, if you have a language option, you can place the compiler on one diskette and the Object Time System (OTS) on another.

If your optional software uses conditional sets of files, you can place each conditional set of files on a separate diskette. Again, using the language option as an example, you could place a compiler and the OTS for systems that have floating-point support on one diskette and place the second set of files for systems without floating-point support on another diskette.

8.1.3 Setting Up Files

You must set up two files before you create an optional software diskette kit: the INSTALL.DAT file and the INS file. The INSTALL.DAT file contains statements that control how the optional software is copied to the fixed disk. The INS file contains the conditional or nonconditional statements for installing and setting up the optional software.

The formatting requirements for setting up the INSTALL.DAT file and the option INS file are as follows:

- Statements in the file must begin in the first column.
- A space must appear after the statement. For INS file statements, a space must also be placed before its argument. Make sure you use the space bar and not the TAB key to insert your spaces.
- Comments must appear on a line that is separate from the statement.

If you do not meet these formatting requirements, an error will occur when the user attempts to install the optional software kit. The following sections explain how to create the INSTALL.DAT and INS files.

8.1.4 Creating the INSTALL.DAT File

The first diskette for each kit must contain a file named INSTALL.DAT, residing in directory [0,0]. The INSTALL.DAT file contains a minimum of three statements that describe the kit media, the method of copying the kit files, and the option INS file name and location.

8.1.4.1 INSTALL.DAT File Statements

The INSTALL.DAT file contains the following statements (the first three statements are required):

- DISKETTES=n
- COPY=YES
  or
  COPY=NO
- OPTION=OPTDESC,[UFD]OPTNAME.INS
- SYSTEM_VERSION=vrsn (optional statement)

These statements are defined as follows:

DISKETTES=n

Specifies the total number, n, of diskettes in the kit you are creating.

COPY=YES
or
COPY=NO

Specifies how the files are copied from the kit onto the fixed disk. You can copy all of the files or you can copy specific files.

The Yes argument specifies that all of the files on the kit are to be copied. You cannot use the Yes argument with FILE and/or BACKUP_SET statements in the option INS file described in Section 8.1.5.

If you are using multiple diskettes with the COPY=YES statement (refer to Example 8–3), you must specify the OPTNAME argument and the sequential number of the diskette in the kit as the volume label for each diskette. This volume label format is necessary because the installation procedure checks each volume label before copying the contents of each diskette to the fixed disk.

The No argument specifies that only selected files are to be copied. You must use the No argument with the FILE and/or BACKUP_SET statements in the option INS file described in Section 8.1.5.

OPTION=OPTDESC,[directory]OPTNAME.INS

Specifies the description of the option to be installed and the complete file specification of the option INS file. You must specify a unique and separate OPTION statement for each software option in the kit. An INSTALL.DAT file may include an unlimited number of OPTION statements.

The OPTDESC argument is the description of the option to be installed. This description can include spaces and any alphanumeric characters. The comma (,) in this statement indicates the end of the option description.

The [directory]OPTNAME.INS argument specifies the directory and the option INS file name on the diskette.

SYSTEM_VERSION=vrsn

Specifies the vrsn argument as the earliest version of the Micro/RSX operating system on which the user can install the option. This is an optional statement.

For diskette kits, the presence of the SYSTEM_VERSION=vrsn statement and the COPY=YES statement in INSTALL.DAT determines if the diskette volume labels will be verified during installation.

- If both the SYSTEM_VERSION=vrsn and COPY=YES statements are present, each diskette volume label will be verified during installation.
- If only the COPY=YES statement is present, the diskette volume labels will not be verified during installation. This allows optional diskette kits that were developed for previous versions of Micro/RSX to install on Micro/RSX Version 4.0.

For tape kits, the SYSTEM_VERSION=vrsn statement does not affect the copying method for optional software kits. Tape kits using the COPY=YES statement must conform to the conditions described in Section 8.2.

8.1.4.2 INSTALL.DAT File Examples

This section provides the following annotated examples of INSTALL.DAT files. These examples illustrate how to utilize the different statements INSTALL.DAT files may include.

- Example 8–1 illustrates a FORTRAN-77 option with the COPY=YES statement, using one diskette.
- Example 8–2 illustrates a FORTRAN-77 option with the COPY=NO statement, using two diskettes.
- Example 8–3 illustrates a FORTRAN-77 option with the COPY=YES statement, using three diskettes.

Example 8–1: Using COPY=YES with One Diskette

```
DISKETTES=1 ❶
COPY=YES ❷
OPTION=FORTRAN SEVENTY-SEVEN,[1,2]F77.INS ❸
SYSTEM_VERSION=4.0 ❹
```

Notes on Example 8–1:

❶ This statement specifies that this kit contains one diskette.

❷ This statement specifies that all the files in the kit are to be copied. Because both the SYSTEM_VERSION=vrsn and COPY=YES statements are present, the diskette volume label will be verified during installation.

❸ This statement specifies the option description (FORTRAN SEVENTY-SEVEN) and the complete file specification of the INS file ([1,2]F77.INS).

❹ This statement specifies that Version 4.0 is the earliest version of the Micro/RSX operating system on which the user can install the option.

Example 8–2: Using COPY=NO with Two Diskettes

```
DISKETTES=2 ❶
COPY=NO ❷
OPTION=FORTRAN SEVENTY-SEVEN,[1,2]F77.INS ❸
SYSTEM_VERSION=3.1 ❹
```

Notes on Example 8–2:

❶ This statement specifies that this kit contains two diskettes.

❷ This statement specifies that only files specified in the INS file are to be copied.

❸ This statement specifies the option description (FORTRAN SEVENTY-SEVEN) and the complete file specification of the INS file ([1,2]F77.INS).

❹ This statement specifies that Version 3.1 is the earliest version of the Micro/RSX operating system on which the user can install the option.

Example 8–3: Using COPY=YES with Three Diskettes

```
DISKETTES=3 ❶
COPY=YES ❷
OPTION=FORTRAN SEVENTY-SEVEN,[1,2]F77.INS ❸
```

Notes on Example 8–3:

❶ This statement specifies that this kit contains three diskettes.

❷ This statement specifies that all the files in the kit are to be copied. Because the SYSTEM_VERSION=vrsn statement is not specified, the diskette volume labels will not be verified during installation. This will allow optional diskette kits that were developed for Micro/RSX Version 1.0 to install on Micro/RSX Version 4.0.

❸ This statement specifies the option description (FORTRAN SEVENTY-SEVEN) and the complete file specification of the INS file ([1,2]F77.INS).

8.1.5 Creating the INS File

For each software option in a kit, there must be one file with the file type of INS. The INS file contains the statements necessary to copy, install, delete, and remove files. The INS file is referenced each time an option is installed or removed, and during subsequent system startups.

The statements in the INS file can be conditional or nonconditional. You use conditional statements to test for specific system features such as floating-point support. The result of the conditional test determines if the statement is executed. Nonconditional statements execute automatically. The following subsections describe the available INS file statements and conditionals.

8.1.5.1 INS File Statements

A list of statements, with their corresponding arguments and qualifiers, that can be used in the option INS file is as follows:

- ! text
- ABORT taskname
- BACKUP_SET volname
- COMMAND cmd
- DELETE filename
- ERROR - errmsg
- FILE filename/KEEP
  /DELETE
- INSTALL filename/TASK
  /LIBRARY
  /COMMON/[NO]WRITEBACK
- INSTALL_PROCEDURE filename
- INSTALL_VERIFICATION filename
- LIBRARY_INSERT filename
- LIBRARY_DELETE modname1:modname2: . . . modname8
- OPTION_VERSION vrsn
- REMOVE taskname
- REMOVE_PROCEDURE filename
- RUN_IMMEDIATE taskname
- RUN_SYSTEM taskname
- RUN_WAIT taskname
- STARTUP_PROCEDURE filename
- SYSTEM_VERSION vrsn (required statement)

A description of each INS file statement follows.

! text

Indicates that the text immediately following the exclamation point (!) is a comment. Placing an exclamation point as the first character on a line in the file allows you to include comments. An exclamation point with no text following it allows you to insert blank lines.

ABORT taskname

Causes the specified task to be aborted if it is active when the user invokes the OPTION command procedure to remove the option

The taskname argument specifies the active task you want to abort.

BACKUP_SET volname

Works in direct conjunction with the COPY=NO statement in the INSTALL.DAT file. It allows you to copy a complete multivolume file to the fixed disk. When you use this statement with diskettes, individual files that are larger than a single diskette (such as a file containing a large compiler) must be created by backing up the multivolume file to the appropriate number of diskettes, using the BACKUP command. A single diskette holds approximately 770 blocks of information. (Refer to the *Micro/RSX System Manager's Guide* for more details on creating multivolume files.)

The volname argument specifies the volume name of the first diskette in the multivolume set.

COMMAND cmd

Allows you to execute DCL commands during the installation procedure and during subsequent system startups. For example, the LOAD xx: command can be executed to load a device driver. This DCL command is executed after the files are copied from the kit and the tasks are installed, but before a startup procedure command file is invoked.

The cmd argument specifies the DCL command to be executed.

DELETE filename

Causes the specified file to be deleted from the fixed disk when the option is removed. This statement can be used for deleting files that are copied during the installation procedure with the COPY=YES or BACKUP_SET statements; or for deleting files that were not copied to the fixed disk during installation, but which must be deleted when an option is removed from the fixed disk.

The filename argument specifies the name of the file to be deleted when an option is removed from the fixed disk.

ERROR - errmsg

Allows you to create a message that indicates rejection of the installation of an option under certain conditions. For example, in the following statement, the absence of the Floating Point Processor (FPP) causes an error message to be generated on the user's terminal:

```
?NOFPP ERROR - F77 requires a floating point processor
```

The syntax of this statement requires you to insert the space-hyphen-space between the ERROR statement and the error message text.

The errmsg argument specifies the text of the error message. Refer to Section 8.1.5.2 for information on conditionals in INS file statements.

FILE filename/KEEP
/DELETE

Works in conjunction with the COPY=NO statement in the INSTALL.DAT file. It specifies which file to copy during the installation of an option. The statement must be respecified for each additional file.

The filename argument specifies the directory and file name for the file you want copied at installation time. If the file resides on a diskette volume other than the one on which the INSTALL.DAT and option INS files reside, you must supply the volume name as the device name in the file specification. For example:

```
FOROTS:[1,1]F77RMS.TSK
```

Do not specify the volume name if the file resides on the first diskette in the kit (that is, the diskette containing the option INS file).

The installation procedure requests each diskette in sequential order. The procedure also sorts and copies all of the specified files on each diskette.

The file may or may not be deleted when the option is removed from the fixed disk, depending on the qualifier that you supplied with the statement. If you do not specify either qualifier, the file will be deleted by default.

The /KEEP qualifier specifies that the file not be deleted when the option is removed from the fixed disk. This qualifier is useful when the option creates data files that should be retained.

The /DELETE qualifier specifies that the file be deleted from the fixed disk when the option is removed. If you specify the /DELETE qualifier, or you do not specify either qualifier, you need not specify the DELETE statement for that file in the option INS file.

INSTALL filename/TASK
/LIBRARY
/COMMON/[NO]WRITEBACK

Specifies the name of the file you want to be installed in the running system. The argument specifies the file directory and file name. Qualifiers specify whether the file being installed is a task image, a read-only common (that is, a library), or a read-write common data area.

When the user invokes the OPTION command procedure to remove an option (for instance, to replace it with a new version), the command procedure issues the DCL command REMOVE for every INSTALL statement that it reads in the INS file.

The filename argument specifies the complete file specification of the file to be installed.

The /TASK qualifier specifies that the file to be installed is a task image.

The /LIBRARY qualifier specifies that the file to be installed is a read-only common (a library).

The /COMMON qualifier specifies that the file to be installed is a read-write common data area. The /NOWRITEBACK qualifier specifies that the common is checkpointed to the system checkpoint file. The /NOWRITEBACK qualifier is the default for the /COMMON qualifier. The /WRITEBACK qualifier specifies that the common is checkpointed to the task image file. Refer to the INSTALL and REMOVE commands in the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2*.

INSTALL_PROCEDURE filename

Enables the installation procedure to execute a specified indirect command file when the option is installed. You may need to use this statement to customize your software environment. An example of this would be to include an indirect command file that creates

device-specific files in your software. The command file is invoked after all files have been copied to the fixed disk, and before any tasks or libraries have been installed. The user may invoke the command file again as a part of the installation procedure after the option has been installed.

The filename argument specifies the directory and file name of the indirect command file to be invoked.

The following global symbols are defined when the installation procedure invokes the command file:

$$CIS Set to true if the system includes the Commercial Instruction Set (CIS)

$$FCS Set to true if File Control Services (FCS) support has been specified

$$FCSD Set to true if you include the FCS version of the OTS modules in the system library ([1,1]SYSLIB.OLB) as the default

$$FPP Set to true if the system includes a Floating Point Processor (FPP)

$$ID Set to true if the system includes a separate instruction and data space option

$$RMS Set to true if Record Management System (RMS-11) support has been specified for this option

$$RMSD Set to true if you include the RMS-11 version of the OTS modules in the system library as the default

$$SLB Set to true if you include OTS modules for the option in the system library

$$SUPR Set to true if the system includes supervisor-mode libraries

INSTALL_VERIFICATION filename

Causes the installation procedure to invoke the specified Installation Verification Procedure (IVP) indirect command file after all other steps in the installation are complete. Entry parameters, such as predefined global variables, are the same as those specified for the installation command file.

When the IVP has completed operation, it should exit with a status indicating success, error, warning, or severe error. (Refer to the exit status codes documented for Indirect in the *Micro/RSX User's Guide, Volume 1.*)

The filename argument specifies the directory and file name for the IVP procedure.

LIBRARY_INSERT filename

Allows you to insert specific object modules into the system library. This statement is helpful with options that have OTS modules (such as FORTRAN-77).

The Librarian Utility Program (LBR) is used to insert the modules into SYSLIB. The object library specified must be in the standard OBJ format.

The filename argument specifies the directory and file name of the object library containing the modules to be inserted into SYSLIB.

Note

DIGITAL recommends that, when possible, you provide each option with a memory-resident OTS library, and avoid inserting modules into SYSLIB.

LIBRARY_DELETE modname1:modname2: . . . modname8

Specifies that when an option is removed, the specified object modules are to be deleted from the system library.

The modname argument specifies the name of the module that is to be deleted from the system library whenever the option is removed. You can specify a maximum of eight modules with each LIBRARY_DELETE statement.

OPTION_VERSION vrsn

Specifies the version number of the optional software kit you create.

The vrsn argument specifies the version number of the kit.

REMOVE taskname

Specifies the task to remove when an option is removed from the fixed disk. You must use the REMOVE statement in the INS file to remove a task, when the file name in the INSTALL statement is different from the installed task name.

If you refer to Example 8–4 you will note that the installed task name for F77.TSK is " . . . F77". In this case, you need the REMOVE statement in the option INS file because the file name used with the INSTALL statement (F77) is different from the name of the installed task (. . . F77).

The taskname argument specifies the name of the installed task you want to remove.

REMOVE_PROCEDURE filename

Enables the installation procedure to execute a specified indirect command file when the option is removed. You can use this statement to disable options. The installation procedure invokes the command file before any tasks are removed and files are deleted.

Entry parameters, such as global variables predefined by the installation procedure, are the same as those specified for the installation command file.

The filename argument specifies the directory and file name of the command file you want to invoke.

RUN_IMMEDIATE taskname

Allows the installation procedure to run an installed task to create the initial environment for the option at installation time and to create the same environment during subsequent system startups. The installation procedure does not wait for the task to exit before continuing (see the RUN_WAIT statement).

The taskname argument specifies the name of the task to be run.

RUN_SYSTEM taskname

Allows the installation procedure to run an installed task at installation time and during subsequent system startups. It is useful for message handlers or despooler tasks that must be active (but not associated with a specific user terminal) when an option is in use.

The taskname argument specifies the name of the task to be run.

RUN_WAIT taskname

Allows the installation procedure to run an installed task to create the initial environment for the option at installation time and to create the same environment during subsequent system startups. The installation procedure will wait for the task to exit or return status before continuing (see RUN_IMMEDIATE).

The taskname argument specifies the name of the task to be run.

STARTUP_PROCEDURE filename

Allows the installation procedure to invoke a specified indirect command file at installation time and during subsequent system startups that create the initial environment for the option. This command file will be invoked by the installation procedure after the files are copied from the kit and the tasks are installed.

Entry parameters, such as global variables predefined by the installation procedure, are the same as those specified for the installation command file.

The filename argument specifies the directory and file name of the command file you want to invoke.

SYSTEM_VERSION vrsn

Specifies the earliest version of the Micro/RSX operating system on which the user can install an option.

The vrsn argument specifies the earliest version number.

Note

This is the only required statement and must be included in every option INS file.

8.1.5.2 Conditionals in INS File Statements

It is sometimes necessary for you to make the installation of some options conditional. For example, an option may be dependent on the presence of the floating-point hardware in order to execute. You can include statements in the option INS file to test for specific system features. For example, the following statement tests for the presence of floating-point hardware before copying the file F77FPP.TSK from the volume F77OTS:

```
?FPP FILE F77OTS:[1,1]F77FPP.TSK
```

You can also negate conditionals, as in the following example:

```
?NOFPP FILE F77OTS:[1,1]F77NOFPP.TSK
```

You can combine conditionals on the same line, as in the following example:

```
?NOFPP?NOCIS FILE F77OTS:[1,1]F77NFPCIS.TSK
```

When an INS file includes conditionals, the installation procedure creates a parameter file. The file name of the parameter file includes the name of your INS file with the file type of PRM. For example, if the name of your INS file is F77.INS, your parameter file name is F77.PRM. The parameter file contains the definitions for the conditionals (such as FCS NO, SYSLIB YES, and so on). The installation procedure uses the parameter file (if there is one) at system-startup

time to perform any conditional installations or to conditionally delete files when an option is removed.

The following conditionals are valid in option INS file statements:

CIS Tests for the presence of the Commercial Instruction Set.

FCS Used with options that support both RMS-11 and FCS to test for FCS support. If you want to specify a record access method, you can instruct the user to invoke the OPTION.CMD command procedure and use the /FULL qualifier to specify the type of support.

FCSDEF Specifies whether the option's OTS routines included in SYSLIB are an FCS version by default.

FPP Tests for the presence of the Floating Point Processor (FPP).

ID Tests for the presence of separate instruction and data space.

RMS Tests for RMS-11 support. (See FCS.)

RMSDEF Specifies whether the option's OTS routines included in SYSLIB are an RMS-11 version by default.

SUPER Tests for the presence of supervisor-mode library support.

SYSLIB Prompts the user to specify whether or not to include OTS routines (or other object library routines) supporting the option in the default system library (SYSLIB). The user's response determines the value for the conditional.

8.1.5.3 INS File Examples

This section provides the following five annotated examples of INS files. They illustrate how to utilize INS file statements and qualifiers frequently specified.

- Example 8–4 illustrates how to set up your files and nonconditional file statements on a single diskette.
- Example 8–5 illustrates how to set up your files and nonconditional file statements on two diskettes.
- Example 8–6 illustrates how to set up your files and conditional and nonconditional file statements on two diskettes.
- Example 8–7 illustrates how to set up your files and conditional file and BACKUP_SET statements on four diskettes.
- Example 8–8 illustrates how to set up your files and conditional file statements for inserting OTS modules into the system library on two diskettes. (Note that this is *not* a recommended procedure. For more information refer to the LIBRARY_INSERT statement in Section 8.1.5.1.)

Example 8–4: Creating an Option on a Single Diskette

```
------------------------------------------------------------
  First Diskette - Labeled  "FORTRAN-77  1 of 1"
                   Volume Name:  F77
------------------------------------------------------------

[0,0]
     INSTALL.DAT

          DISKETTES=1                          )
          COPY=YES                             ) ❶
          OPTION=FORTRAN-77,[1,2]F77.INS       )
          SYSTEM_VERSION=4.0

[1,1]
     F77OTS.STB
     F77OTS.TSK
[1,2]
     F77.INS

          !
          ! FORTRAN-77 option parameter file
          !
          INSTALL [3,54]F77/TASK               )
          INSTALL [1,1]F77OTS/LIBRARY          ) ❷
          !
          REMOVE ...F77  ❸
          !
          DELETE [1,1]F77*.*                   )
          DELETE [1,2]F77COM.MSG,FORTRAN.HLP   ) ❹
          DELETE [3,54]F77.TSK                 )
          !
          SYSTEM_VERSION 4.0  ❺
     FORTRAN.HLP
     F77COM.MSG

[3,54]
     F77.TSK

------------------------------------------------------------
```

Notes on Example 8–4:

❶ The first three statements must be specified in the INSTALL.DAT file.

❷ These statements specify the task image files installed on the running system when the user installs the option.

❸ This statement specifies the task image file removed from the running system when the user removes the option from the fixed disk.

❹ These statements specify the files deleted when the user removes the option from the fixed disk.

❺ This statement specifies the earliest version of the Micro/RSX operating system on which the user can install the F77 option.

Example 8–5: Creating an Option on Multiple Diskettes

```
------------------------------------------------------------
  First Diskette - Labeled  "FORTRAN-77  1 of 2"
                   Volume Name:  F77
------------------------------------------------------------

[0,0]
     INSTALL.DAT

          DISKETTES=2                                  )
          COPY=NO                                      ) ❶
          OPTION=FORTRAN-77,[1,2]F77.INS               )
          SYSTEM_VERSION=1.1

[1,2]
     F77.INS
          !
          ! FORTRAN-77 option installation file
          !
          FILE [3,54]F77.TSK/DELETE                    )
          FILE F77OTS:[1,1]F77OTS.*/DELETE             ) ❷
          FILE [1,2]F77COM.MSG                         )
          !
          FILE [1,2]FORTRAN.HLP/KEEP  ❸
          !
          INSTALL [3,54]F77/TASK                       )
          INSTALL [1,1]F77OTS/LIBRARY                  ) ❹
          !
          REMOVE ...F77  ❺
          !
          SYSTEM_VERSION 1.1  ❻

     FORTRAN.HLP
     F77COM.MSG

[3,54]
     F77.TSK

------------------------------------------------------------
  Second Diskette - Labeled  "FORTRAN-77  2 of 2"
                    Volume Name:  F77OTS
------------------------------------------------------------

[1,1]
     F77OTS.STB
     F77OTS.TSK
------------------------------------------------------------
```

Notes on Example 8–5:

❶ The first three statements must be specified in the INSTALL.DAT file.

❷ These statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes the option from the disk.

❸ This statement specifies the file that is copied to the fixed disk when the user installs the option, and which is saved when the user removes the option from the disk.

❹ These statements specify the task image files installed on the running system when the user installs the option.

❺ This statement specifies the task image file removed from the running system when the user removes the option from the fixed disk.

❻ This statement specifies the earliest version of the Micro/RSX operating system on which the user can install the F77 option.

Example 8–6: Using Conditionals in an INS File

```
------------------------------------------------------------
  First Diskette - Labeled  "FORTRAN-77  1 of 2"
                   Volume Name:  F77
------------------------------------------------------------

[0,0]
     INSTALL.DAT

          DISKETTES=2                                  )
          COPY=NO                                      } ❶
          OPTION=FORTRAN-77,[1,2]F77.INS               )
          SYSTEM_VERSION=4.0

[1,2]
     F77.INS
          !
          ! FORTRAN-77 option installation file
          !
          FILE [3,54]F77.TSK/DELETE                    )
          FILE [1,2]F77COM.MSG                         } ❷
          FILE [1,2]FORTRAN.HLP                        )
          !
          ?FCS FILE F77OTS:[1,1]F77FCS.TSK/DELETE      )
          ?FCS FILE F77OTS:[1,1]F77FCS.STB/DELETE      }
          ?RMS FILE F77OTS:[1,1]F77RMS.TSK/DELETE      } ❸
          ?RMS FILE F77OTS:[1,1]F77RMS.STB/DELETE      )
          !
          INSTALL [3,54]F77/TASK
          ?FCS INSTALL [1,1]F77FCS/LIBRARY             } ❹
          ?RMS INSTALL [1,1]F77RMS/LIBRARY             )
          !
          REMOVE ...F77                                } ❺
          REMOVE F77OTS                                )
          !
          SYSTEM_VERSION 4.0  ❻

     FORTRAN.HLP
     F77COM.MSG

[3,54]
     F77.TSK

------------------------------------------------------------
  Second Diskette - Labeled  "FORTRAN-77  2 of 2"
                    Volume Name:  F77OTS
------------------------------------------------------------
[1,1]
     F77FCS.STB
     F77FCS.TSK
     F77RMS.STB
     F77RMS.TSK
------------------------------------------------------------
```

Notes on Example 8–6:

❶ The first three statements must be specified in the INSTALL.DAT file.

❷ These statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes the option from the disk.

❸ These conditional statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes the option from the disk.

❹ These conditional statements specify the task image files installed on the running system when the user installs the option.

❺ These statements specify the task images removed from the running system when user removes the option from the fixed disk.

❻ This statement specifies the earliest version of the Micro/RSX operating system on which the user can install the F77 option.

Example 8–7: Using Conditionals and Multivolume Files

```
------------------------------------------------------------
  First Diskette - Labeled  "FORTRAN-77  1 of 4"
                   Volume Name:  F77
------------------------------------------------------------

[0,0]
     INSTALL.DAT
          DISKETTES=2                          )
          COPY=NO                              } ❶
          OPTION=FORTRAN-77,[1,2]F77.INS       )
          SYSTEM_VERSION=1.1
[1,2]
     F77.INS
          !
          ! FORTRAN-77 installation parameter file
          !
          ?FCS BACKUP_SET F77CMPFCS            } ❷
          ?RMS BACKUP_SET F77CMPRMS            )
          !
          FILE [1,2]F77COM.MSG                 } ❸
          FILE [1,2]FORTRAN.HLP                )
          !
          ?FCS FILE F77OTS:F77FCS.TSK/DELETE   )
          ?FCS FILE F77OTS:F77FCS.STB/DELETE   } ❹
          ?RMS FILE F77OTS:F77RMS.TSK/DELETE   }
          ?RMS FILE F77OTS:F77RMS.STB/DELETE   )
          !
          INSTALL [3,54]F77/TASK
          ?FCS INSTALL [1,1]F77FCS/LIBRARY     } ❺
          ?RMS INSTALL [1,1]F77RMS/LIBRARY     )
          !
          DELETE [3,54]F77.TSK  ❻
          !
          REMOVE ...F77                        } ❼
          REMOVE F77OTS                        )
          !
          SYSTEM_VERSION 1.1  ❽
     FORTRAN.HLP
     F77COM.MSG

------------------------------------------------------------
  Second Diskette - Labeled  "FORTRAN-77  2 of 4"
                    Volume name:  F77OTS
------------------------------------------------------------

[1,1]
     F77FCS.STB
     F77FCS.TSK
     F77RMS.STB
     F77RMS.TSK
```

(Continued on next page)

Example 8–7 (Cont.): Using Conditionals and Multivolume Files

```
------------------------------------------------------------
  Third Diskette - Labeled  "FORTRAN-77  3 of 4"
                   Volume Name:  F77CMPFCS
------------------------------------------------------------
[0,0]BACKUP.SYS
   [3,54]F77.TSK

------------------------------------------------------------
  Fourth Diskette - Labeled  "FORTRAN-77  4 of 4"
                    Volume Name:  F77CMPRMS
------------------------------------------------------------
[0,0]BACKUP.SYS
   [3,54]F77.TSK

------------------------------------------------------------
```

Notes on Example 8–7:

❶ The first three statements must be specified in the INSTALL.DAT file.

❷ These conditional statements specify the backup set copied to the fixed disk when the user installs the option.

❸ These statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes the option from the disk.

❹ These conditional statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes option from the disk.

❺ These conditional statements specify the task image files installed on the running system when the user installs the option.

❻ This statement specifies the name of the backup set deleted when the user removes the option from the fixed disk.

❼ These statements specify the task images removed from the running system when the user removes the option from the fixed disk.

❽ This statement specifies the earliest version of the Micro/RSX operating system on which the user can install the F77 option.

Example 8–8: Inserting Modules into SYSLIB

```
------------------------------------------------------------
  First Diskette - Labeled  "FORTRAN-77  1 of 2"
                   Volume Name:  F77
------------------------------------------------------------

[0,0]
     INSTALL.DAT

          DISKETTES=2                                  )
          COPY=NO                                      } ❶
          OPTION=FORTRAN-77,[1,2]F77.INS               )
          SYSTEM_VERSION=4.0

[1,2]
     F77.INS
          !
          ! FORTRAN-77 installation parameter file
          !
          FILE [1,2]F77*.*/DELETE                      }❷
          FILE [3,54]F77.TSK/DELETE                    )
          !
          ?SYSLIB FILE F77OTS:[11,36]F77OTS.OBJ/DELETE          )
          ?NOSYSLIB FILE F77OTS:[1,1]F77OTS.OLB/DELETE          |
          !                                                     |
          ?FCSDEF FILE F77OTS:[11,36]F77OTSFCS.OBJ/DELETE       |
          ?NOSYSLIB?FCS FILE F77OTS:[1,1]F77OTSFCS.OLB/DELETE   |
          !                                                     } ❸
          ?RMSDEF FILE F77OTS:[11,36]F77OTSFCS.OBJ/DELETE       |
          ?NOSYSLIB?RMS FILE F77OTS:[1,1]F77OTSRMS.OLB/DELETE   |
          !                                                     |
          ?SYSLIB LIBRARY_INSERT [11,36]F77OTS.OBJ     )        |
          ?FCSDEF LIBRARY_INSERT [11,36]F77OTSFCS.OBJ  } ❹      |
          ?RMSDEF LIBRARY_INSERT [11,36]F77OTSRMS.OBJ  )        )
          !
          LIBRARY_DELETE F77OTS  ❺
          !
          INSTALL [3,54]F77  ❻
          !
          REMOVE ...F77  ❼
          !
          SYSTEM_VERSION 4.0  ❽
     FORTRAN.HLP
     F77COM.MSG

[3,54]
     F77.TSK

[11,36]
     F77TST.FTN
```

(Continued on next page)

Example 8–8 (Cont.): Inserting Modules into SYSLIB

```
------------------------------------------------------------
  Second Diskette - Labeled  "FORTRAN-77  2 of 2"
                    Volume name:  F77OTS
------------------------------------------------------------

[1,1]
     F77OTS.OLB
     F77OTSFCS.OLB
     F77OTSRMS.OLB
[11,36]
     F77OTS.OBJ
     F77OTSFCS.OBJ
     F77OTSRMS.OBJ
------------------------------------------------------------
```

Notes on Example 8–8:

❶ The first three statements must be specified in the INSTALL.DAT file.

❷ These statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes the option from the disk.

❸ These conditional statements specify the files that are copied to the fixed disk when the user installs the option, and which are deleted when the user removes the option from the disk.

❹ These conditional statements specify the modules inserted into the system library when the user installs the option.

❺ This statement specifies the module deleted from the system library when the user removes the option from the fixed disk.

❻ This statement specifies the task image file installed on the running system when the user installs the option.

❼ This statement specifies the task image removed from the running system when the user removes the option from the fixed disk.

❽ This statement specifies the earliest version of the Micro/RSX operating system on which the user can install the F77 option.

8.1.6 Creating a Diskette Kit

To create a diskette kit, you copy your files from your diskettes or from the fixed disk to blank diskettes. If you have single volume and multivolume files, it is recommended that you copy your single volume files before you copy your multivolume files.

- For information on copying single volume files to diskettes, refer to Section 8.1.6.1.
- For information on copying multivolume files, refer to Section 8.1.6.2.

8.1.6.1 Copying Single Volume Files

This section provides the steps you should follow to copy single volume files to diskettes.

1. Insert a blank diskette in DU1 and allocate that device. Type the following:

```
$ ALLOCATE DU1:
```

2. Mount DU1 and specify the /FOREIGN qualifier. Type the following:

```
$ MOUNT DU1:/FOREIGN
```

3. Analyze the blank diskette in DU1 to create a bad block file on the diskette. Type the following:

```
$ ANALYZE/MEDIA DU1:
```

4. Initialize the blank diskette in DU1 and specify the volume label for your option. Type the following:

```
$ INITIALIZE DU1:volumelabel
```

5. Dismount DU1. Then, remount DU1 and specify the new volume label. Type the following:

```
$ DISMOUNT DU1:
$ MOUNT DU1:volumelabel
```

   - If your optional software files are on the fixed disk, proceed to step 8.
   - If your optional software files are on diskettes, continue with step 6.

6. Insert the diskette in DU2 and allocate that device. Then, mount DU2 and specify the volume label for your diskette. Type the following:

```
$ ALLOCATE DU2:
$ MOUNT DU2:volumelabel
```

7. Set your default directory and create the directory on the diskette in DU1. Then, copy the files from the diskette in DU2 to the diskette in DU1. Type the following:

```
$ SET DEFAULT [directory]
$ CREATE/DIRECTORY DU1:[directory]
$ COPY DU2:[directory]files DU1:[directory]
```

Parameters

directory

Specifies the directory containing the optional software files.

files

Specifies the wildcard specification or the list of file names you want to copy from the diskette in DU2 to the diskette in DU1.

Repeat this step for each directory you want to copy to the diskette in DU1. When you have copied all the directories, proceed to step 9.

8. If your optional software files are on the fixed disk, set your default directory and create the directory on the diskette in DU1. Then, copy the files from DU0 to the diskette in DU1. Type the following:

```
$ SET DEFAULT [directory]
$ CREATE/DIRECTORY DU1:[directory]
$ COPY DUO:[directory]files DU1:[directory]
```

Parameters

directory

Specifies the directory containing the optional software files.

files

Specifies the wildcard specification or the list of file names you want to copy from DU0 to the diskette in DU1.

Repeat this step for each directory you want to copy to the diskette in DU1.

9. Dismount DU1 and remove the diskette from that device. Type the following:

```
$ DISMOUNT DU1:
```

If your optional software files were on diskettes, it is necessary to dismount DU2 and remove the diskette from that device. Type the following:

```
$ DISMOUNT DU2:
```

Repeat steps 2 to 9 for each diskette containing single volume files in your optional software kit.

10. Deallocate DU1. Type the following:

```
$ DEALLOCATE DU1:
```

If your optional software files were on diskettes, it is necessary to deallocate DU2. Type the following:

```
$ DEALLOCATE DU2:
```

Now that you have copied all your single volume files to diskettes, proceed as follows:

- If you must copy multivolume files to complete your diskette kit, proceed to Section 8.1.6.2.
- If your diskette kit is complete (you do not have multivolume files to copy) and you are *not* creating a tape package, proceed to Section 8.3 to complete your diskette package.
- If your diskette kit is complete (you do not have multivolume files to copy) and you *are* creating a tape package, proceed to Section 8.2.

8.1.6.2 Copying Multivolume Files

This section provides the steps you should follow to copy multivolume files from the fixed disk to diskettes.

1. Insert a blank diskette in DU1 and allocate that device. Type the following:

   ```
   $ ALLOCATE DU1:
   ```

2. Mount DU1 and specify the /FOREIGN qualifier. Type the following:

   ```
   $ MOUNT DU1:/FOREIGN
   ```

3. Analyze the blank diskette in DU1 to create a bad block file on the diskette. Type the following:

   ```
   $ ANALYZE/MEDIA DU1:
   ```

4. Dismount DU1 and remove the diskette from that device. Type the following:

   ```
   $ DISMOUNT DU1:
   ```

   Repeat steps 2 to 4 for each diskette in the multivolume backup sets.

5. Insert an analyzed blank diskette in DU1, remount that device, and specify the /FOREIGN qualifier. Type the following:

   ```
   $ MOUNT DU1:/FOREIGN
   ```

6. Using the BACKUP command and specifying the /SAVE and /VERIFY qualifiers, copy the multivolume file from DU0 to the diskette in DU1. Type the following:

   ```
   $ BACKUP/SAVE:name/VERIFY DU0:file DU1:
   ```

   #### Parameters

   name

   Specifies the name used in the option INS file BACKUP_SET statement.

   file

   Specifies the directory and file name of the multivolume file on DU0 that you want to copy to the diskette in DU1.

   Repeat this step for each additional multivolume file in your kit to copy from the fixed disk to diskettes.

7. Dismount DU1 and deallocate that device. Then, remove the diskette from DU1. Type the following:

   ```
   $ DISMOUNT DU1:
   $ DEALLOCATE DU1:
   ```

Now that you have copied all your multivolume files and completed your diskette kit, proceed as follows:

- If your diskette kit is complete and you are creating a tape package, proceed to Section 8.2.
- If your diskette kit is complete and you are *not* creating a tape package, proceed to Section 8.3 to complete your diskette package.

8.2 Preparing to Create a Tape Package

This section describes the resources you need and the steps you must follow to create an optional software tape kit (which combined with your documentation completes the tape package). The resources you need are as follows:

- A Micro/RSX Version 4.0 operating system
- A copy of the optional software kit on the fixed disk or on RX50 diskettes
- A TK50 tape drive
- A TK50 tape cartridge

If you are using a MicroPDP-11 with two RX50 diskette drives, you also need blank RX50 diskettes.

Note

You must create the optional software kit on diskette (refer to Section 8.1) before you can begin creating the tape package.

Once you have the necessary resources, you must complete the following steps to create an optional software tape package:

1. Create the template INS file.
2. Optimize the template INS file.
3. Edit the INSTALL.DAT file and the template INS file.
4. Organize the optional software files.
5. Create the tape kit by copying the files from diskette or diskettes or the fixed disk.
6. Document the installation procedure (refer to Section 8.3).

The following sections describe each step in detail, beginning with Section 8.2.1.

If the corresponding optional software diskette kit INSTALL.DAT file contains the COPY=YES statement and the OPTION.INS file does not contain FILE or BACKUP_SET statements, you do not need to create a template tape OPTION.INS file with the TAPEINS.CMD procedure. TAPEINS.CMD will return an error message if the input diskette OPTION.INS file does not contain FILE or BACKUP_SET statements.

To create an optional software tape kit that uses the COPY=YES statement, you must copy the INSTALL.DAT and OPTION.INS files to a backup set named INSTALL. All other kit files must be copied to one backup set named after the file name portion of the OPTION.INS file.

8.2.1 Creating the Template INS File

The following procedure organizes the FILE and BACKUP_SET statements in the diskette option INS file, based on the conditional status of these statements. This enables you to copy your files into the correct backup sets on the tape kit. The following procedure also creates the template INS file you will use as a guide when you complete the tape kit. A message on your terminal tells you when you have completed the procedure. If a problem occurs during the procedure, refer to Section 8.4 for corrective action.

Note

This procedure assumes you are using a valid diskette option INS file as input. No syntax checks are made by this procedure. All FILE and BACKUP_SET statements must be placed before other option INS file statements.

- If your INS file is on a diskette, follow steps 1 to 8 to create the template INS file.
- If your INS file is on the fixed disk, follow steps 3 to 8 to create the template INS file.

1. Insert in DU1 the first diskette of the kit; this diskette contains the diskette option INS file.
2. Allocate DU1, mount the diskette in DU1, and specify the volume label for your optional software. Type the following:

```
$ ALLOCATE DU1:
$ MOUNT DU1:volumelabel
```

3. Invoke the indirect command file by typing the following:

```
$ @TAPEINS
```

The indirect command file displays the following message and prompt:

```
*     This command procedure accepts as input a valid
*     Micro/RSX Optional Software diskette option .INS
*     file and creates a template tape option .INS file.
*     This template .INS file is used as guide in producing
*     a Micro/RSX Optional Software Tape Kit.
*
*     DD-MMM-YY      HH:MM:SS
*
*     Enter the file name and type of the input diskette
*     option .INS file.  Include the device and directory
*     if different from the current default.
*
$* Diskette option .INS file?[s]:
```

4. Type the file name and file type of the input file; that is, the diskette option INS file. Include the device and directory if different from your current default device and directory. Your response should be similar to the following:

```
DU1:[1,2]F77.INS
```

The indirect command file then displays the following prompt:

```
*     Enter the device and directory to create the
*     template tape option .INS file on.
*     Press only the RETURN key to create the file on the
*     default device and in the default directory.
*
$* Device and directory of the tape option .INS file?[s]:
```

5. Press only the RETURN key if you want to create the output file on your default device and in your default directory. If you want to create the output file on a different device and directory, specify this device and directory first and then press the RETURN key. The TAPEINS.CMD procedure determines the output file name and file type. Your response should be similar to the following:

   ```
   DU0:[USER]
   ```

   The indirect command file reads the diskette option INS file (F77.INS) statements and creates a template tape option INS file (TF77.INS). Messages similar to the following are displayed:

   ```
   *     Reading FILE and BACKUP_SET statements from
   *     DU1:[1,2]F77.INS.
   *
   *     Writing new BACKUP_SET and DELETE statements to
   *     DU0:[USER]TF77.INS.
   *
   *     Copying the remaining statements from
   *     DU1:[1,2]F77.INS to DU0:[USER]TF77.INS.
   ```

   After all the INS file statements have been processed successfully, a message similar to the following appears:

   ```
   *     The TAPEINS.CMD procedure completed successfully.
   *
   *     Your template tape option .INS file is
   *     DU0:[USER]TF77.INS.
   *
   *     DD-MMM-YY       HH:MM:SS
   ```

When the TAPEINS.CMD procedure completes successfully, it creates a template INS file. Use this file as a guide when you create the tape option INS file. Proceed to the following section to determine if you can optimize the template INS file.

8.2.2 Optimizing Your Template INS File

You can optimize your template INS file by merging and/or separating your BACKUP_SET statements. Merging your BACKUP_SET statements allows the user to install the tape kit in less time.

Note

You must separate your BACKUP_SET statements if any BACKUP_SET statement references more than 16 file specifications (the maximum number allowed by the BACKUP command). You may use wildcards in the directory, file name, or file type in your file specification. Using wildcards reduces the file specifications referenced by your BACKUP_SET statement.

Examine your template INS file to determine if you can optimize it. If your template INS file meets either of the conditions, proceed to the specified section or sections, as follows:

- If the template INS file contains more than one BACKUP_SET statement with the same conditional status, proceed to Section 8.2.2.1.
- If a BACKUP_SET statement references more than 16 file specifications, proceed to Section 8.2.2.2.

If you determine that it is not necessary to optimize your template INS file, proceed to Section 8.2.3.

8.2.2.1 Merging BACKUP_SET Statements

This section describes how you merge BACKUP_SET statements. The following procedure is recommended if you have two or more BACKUP_SET statements in your template INS file that contain the same conditional status:

1. If your files are on diskettes, use the COPY command to copy all the files listed in the backup sets to diskette or diskettes. Each diskette of files will later be copied to a backup set on the tape.
2. If your files are on the fixed disk, print a hardcopy list of the file arrangement in the template INS file. Combine the file list from each group of BACKUP_SET statements that contain the same conditional status. Use this list as a guide when you copy your files to the tape.
3. Edit the template INS file. Delete the BACKUP_SET statements with the same conditional status, and replace the statements with one BACKUP_SET statement that has the same conditional status for each group of files to be copied to a backup set on the tape.

If you must also separate your BACKUP_SET statements, proceed to Section 8.2.2.2; otherwise, proceed to Section 8.2.3.

8.2.2.2 Separating BACKUP_SET Statements

This section describes how to separate BACKUP_SET statements in the template INS file. You must separate a BACKUP_SET statement if the BACKUP_SET statement references more than 16 file specifications on the fixed disk or more than 770 blocks on the diskette.

If your files are on diskettes, follow steps 1 to 4 to separate your BACKUP_SET statements.

If your files are on the fixed disk, print a hardcopy list of the files contained in each group to be copied to a backup set on the tape. Separate the files referenced in the backup set list into groups of 16 file specifications. Proceed to step 4 to edit the template INS file.

1. Identify the total number of file specifications with the same conditional status from the files listed in the template INS file.
2. Use the COPY command to copy these files to a blank diskette.
3. If you cannot fit the balance of your files on one diskette, you must repeat step 2 to create an additional group of files on another diskette to copy to the tape as a separate backup set. Each additional group of files requires a unique backup set name and BACKUP_SET statement.

4. Edit the template INS file to create BACKUP_SET statements that reference the new groups of files on diskettes or fixed disk. Each BACKUP_SET statement and the files referenced by the statement must have the same conditional status.

When you complete this procedure, proceed to Section 8.2.3 to edit your files.

8.2.3 Editing INSTALL.DAT and Template INS Files

You must edit the INSTALL.DAT file before continuing to create your tape kit. The template INS file has several sections you may also need to edit. Using EDT to edit your files, refer to the following list for editing instructions:

1. Delete the statement DISKETTE=n in the INSTALL.DAT file and replace it with the statement TAPE=1.
2. Edit the INSTALL.DAT file if you must copy more than one diskette option to the tape kit. Include the OPTION=OPTDESC,[directory]OPTNAME.INS statement from the INSTALL.DAT file of each option you must copy to the tape kit.
3. Delete unnecessary comments created by the TAPEINS command procedure in the template INS file. You may replace these comments with others that are appropriate for your specific optional software.
4. Replace generic backup set names created by the TAPEINS command procedure in the template INS file with names that specify your optional software.
5. Rename the template INS file and replace it with the correct name by using the RENAME command. For example, TF77.INS would become F77.INS.

When your editing is complete, proceed to Section 8.2.4 to organize your files.

8.2.4 Organizing the Optional Software Files

You can organize files that have not been optimized before you copy the files to tape. The lists of files in the template INS file specify how your files are to be copied to tape.

- If your files are on diskettes, proceed to Section 8.2.4.1.
- If your files are on the fixed disk, proceed to Section 8.2.4.2.

8.2.4.1 Organizing Files Using Diskettes

Follow these steps to organize your files on blank diskettes:

1. Insert a blank diskette in DU1 and allocate that device. Type the following:

```
$ ALLOCATE DU1:
```

2. Insert the diskette containing your new INSTALL.DAT and option INS files in DU2 and allocate that device. Type the following:

```
$ ALLOCATE DU2:
```

3. Mount DU2 and specify the volume label for the diskette. Then, mount DU1 and specify the /FOREIGN qualifier. Type the following:

```
$ MOUNT DU2:volumelabel
$ MOUNT DU1:/FOREIGN
```

4. Analyze the blank diskette in DU1 to create a bad block file on the diskette. Type the following:

```
$ ANALYZE/MEDIA DU1:
```

5. Initialize the diskette in DU1 and specify the volume label INSTALL. Dismount DU1. Then, remount DU1 and specify the volume label INSTALL. Type the following:

```
$ INITIALIZE DU1:INSTALL
$ DISMOUNT DU1:
$ MOUNT DU1:INSTALL
```

6. If the directory of the option INS files are not on the diskette with the volume label INSTALL in DU1, type the following to create this directory:

```
$ CREATE/DIRECTORY DU1:[directory]
```

7. Copy the INSTALL.DAT file and the option INS file, created in Section 8.2.3, from the diskette in DU2 to the diskette (with the volume label INSTALL) in DU1. Type the following:

```
$ COPY DU2:[0,0]INSTALL.DAT DU1:[0,0]INSTALL.DAT
$ COPY DU2:filespec DU:filespec
```

Parameter

filespec

Specifies the directory and file name of the option INS file.

8. Dismount DU2 and remove the diskette from that device. Type the following:

```
$ DISMOUNT DU2:
```

- If you are copying only one diskette kit to the tape kit, proceed to step 12.
- If you are copying more than one diskette kit to the tape kit, you must complete steps 9 to 11 before proceeding to step 12.

9. Insert and mount the first diskette for each diskette kit in DU2.

```
$ MOUNT DU2:volumelabel
```

10. Copy the option INS file from the diskette in DU2 to the diskette (with the volume label INSTALL) in DU1. Type the following:

```
$ COPY DU2:filespec DU1:filespec
```

Parameter

filespec

Specifies the directory and file name of the option INS file.

11. Dismount DU2 and remove the diskette from that device. Type the following:

```
$ DISMOUNT DU1:
```

    Repeat steps 9 to 11 for each diskette kit you want to copy to the tape kit.

12. Dismount DU1 and remove the diskette (with the volume label INSTALL) from that device. Type the following:

```
$ DISMOUNT DU1:
```

13. Using the COPY command, copy each group of your optional software files to diskettes according to the tape INS file backup set lists.

14. Deallocate DU2 and DU1. Type the following:

```
$ DEALLOCATE DU2:
$ DEALLOCATE DU1:
```

    Once you have completed the previous steps for each of your options, proceed to Section 8.2.5 for the steps you must follow to copy your files to tape.

8.2.4.2 Organizing Files on the Fixed Disk

This section describes how to organize your files on the fixed disk. You must complete the following procedure for any files you did not previously optimize.

1. Print a hardcopy list of the template INS file.
2. Determine which files are located in each backup set by referencing the file arrangement in the template INS file BACKUP_SET statements. Use the list as a guide when you copy the files to tape.

When you complete this procedure for each option, you must copy to tape. Proceed to Section 8.2.5 to copy your optional software files to tape.

8.2.5 Creating the Tape Kit

The physical location of each group of files referenced by your BACKUP_SET statements will determine how you copy these files to the tape kit.

- To copy a group of files from diskettes to tape, proceed to Section 8.2.5.1.
- To copy a group of files from fixed disk to tape, proceed to Section 8.2.5.2.

8.2.5.1 Copying Files from Diskettes to Tape

If you have a MicroPDP-11 with two RX50 diskette drives, your option files should be organized on diskettes. To copy each group of option files on diskettes to tape, use the following steps:

1. Insert in DU2 the diskette with the volume label INSTALL. Allocate DU2. Then, mount DU2 and specify the /FOREIGN qualifier. Type the following:

```
$ ALLOCATE DU2:
$ MOUNT DU2:/FOREIGN
```

2. Load the TK50 tape cartridge in MU0 and allocate that device. Then, mount MU0 specifying the /FOREIGN qualifier. Type the following:

```
$ ALLOCATE MU0:
$ MOUNT MU0:/FOREIGN
```

3. Copy the diskette with the volume label INSTALL from DU2 to the INSTALL backup set on MU0. Type the following:

```
$ BACKUP/REWIND/VERIFY/SAVE:INSTALL DU2: MU0:
```

4. Dismount DU2 and remove the diskette with the volume label INSTALL from that device. Type the following:

```
$ DISMOUNT DU2:
```

5. Insert the diskette containing the first set of option kit files in DU2. Mount DU2 and specify the /FOREIGN qualifier. Type the following:

```
$ MOUNT DU2:/FOREIGN
```

6. Copy the files from the diskette in DU2 to MU0. Type the following:

```
$ BACKUP/APPEND/VERIFY/SAVE:name DU2: MU0:
```

Parameter

name

Specifies the name in the option INS file for the BACKUP_SET statement.

7. Dismount DU2 and remove the diskette from that device. Type the following:

```
$ DISMOUNT DU2:
```

Repeat steps 5 to 7 for each set of files on diskette that you must copy to a backup set on the tape kit. Then proceed to step 8.

8. Dismount MU0, deallocate MU0, and remove the tape from MU0. Type the following:

```
$ DISMOUNT MU0:
$ DEALLOCATE MU0:
```

9. Deallocate DU2. Type the following:

```
$ DEALLOCATE DU2:
```

- If additional groups of backup set files are contained on the fixed disk, proceed to Section 8.2.5.2 to complete copying your files to the tape.
- If you have no other groups of backup set files to copy, proceed to Section 8.3 to document your tape package.

8.2.5.2 Copying Files from Fixed Disk to Tape

If you have a MicroPDP-11 without RX50 diskette drives, your option kit files should be organized on the fixed disk. Use the following steps to copy each set of files from the fixed disk to tape:

1. Load the TK50 tape in MU0 and allocate that device. Then, mount MU0 and specify the /FOREIGN qualifier. Type the following:

```
$ ALLOCATE MU0:
$ MOUNT MU0:/FOREIGN
```

2. Copy the INSTALL.DAT file and the option INS file or files from DU0 to MU0, and specify INSTALL as the name of the backup set. Type the following:

```
$ BAC/REW/VER/SAV:INSTALL/MOU DU0:[0,0]INSTALL.DAT,files MU0:
```

Parameter

files

Indicates the wildcard specification or the option INS file names on DU0 you want to copy into the backup set on tape.

3. For the first set of files in the kit, copy the files from DU0 to MU0. Type the following:

```
$ BACKUP/APPEND/VERIFY/SAVE:name/MOUNT DU0:files MU0:
```

Parameters

name

Specifies the name used in the option INS file for the corresponding BACKUP_SET statement.

files

Indicates the wildcard specification or the file names on DU0 you want to copy into the specified backup set.

Repeat this step for each set of files you want to copy from the fixed disk to a backup set on the tape.

4. Dismount and deallocate MU0. Then, remove the tape from MU0. Type the following:

```
$ DISMOUNT MU0:
$ DEALLOCATE MU0:
```

- If additional groups of backup set files are contained on diskettes, return to Section 8.2.5.1 to complete copying your files to the tape.
- If you have no other groups of backup set files to copy, proceed to Section 8.3 to document your tape package.

8.3 Documenting the Installation Procedure

A complete diskette or tape package includes the software kit and installation documentation. You can complete your package by using the appropriate prototype installation guides included in this manual. Appendix B contains the prototype installation guide for installing a diskette kit. Appendix C contains the prototype installation guide for installing a tape kit. By inserting option-specific information, these guides can be used for the installation procedure for an optional software kit.

Micro/RSX software installation guides use these prototypes to document the installation procedure. (Refer to the *Micro/RSX Advanced Programmer's Kit Installation Guide for Diskettes* for an example of documentation produced using the prototype in Appendix B. Refer to the *Micro/RSX Advanced Programmer's Kit Installation Guide for Tape* for an example of documentation produced using the prototype in Appendix C.) These prototypes provide a brief introduction to the installation procedure, describe the required software, give step-by-step installation instructions, and provide troubleshooting information.

To produce your installation guide you must insert into the prototype information specific to your optional software kit. Symbols enclosed in angle brackets (<>) indicate where you should make these insertions within the prototype. When you produce your installation guide, make sure you delete these angle bracket symbols as you insert your option-specific information. Refer to Table 8–1 for descriptions of the angle bracket symbols used in the prototypes.

Table 8–1: Angle Bracket Symbols

Symbol	Description
<PRODUCT_NAME>	Specifies the product name of the software being installed.
<NUMBER_OF_MINUTES>	Specifies the number of minutes required to install the software.
<NUMBER_OF_DISKETTES>	Specifies the number of diskettes used to install the software.
<NUMBER_OF_TAPES>	Specifies the number of tapes used to install the software.
<LABEL_NAME>	Specifies the label name of the diskette or tape.
<BOX_LABEL>	Specifies the label name of the box containing the tape or tapes.
<SENTENCE_DESCRIBING _CONTENTS_OF_DISKETTE>	Describes the contents of the diskette used to install the software.
<SENTENCE_DESCRIBING _CONTENTS_OF_TAPE>	Describes the contents of the tape used to install the software.
<MANUAL_TITLE>	Specifies the title of a manual that will direct the user to more information on the software.

Table 8–1 (Cont.): Angle Bracket Symbols

Symbol	Description
<IVP_MESSAGE>	Specifies the message produced upon successful completion of the Installation Verification Procedure (IVP).
<ACCOUNT>	Specifies the location of the file copied to the fixed disk during the installation of the software.
<FILE>	Specifies the file copied to the fixed disk during the installation of the software.
<DESCRIPTION>	Specifies the description of the file copied to the fixed disk during the installation of the software.
<PARA_DESCRIBING _CONDITIONALS_IF_APPLICABLE>	Describes conditionalizing the option during the installation of the software.

8.4 Error Messages

This section lists the error messages produced by the indirect command procedure in Section 8.2.1. Explanations and suggested user action accompany each error message. Refer to the *Micro/RSX User's Guide, Volume 1* and *Micro/RSX User's Guide, Volume 2* for corrective action on other messages produced during the development procedure of your optional diskette or tape kit.

Following is a list of error messages returned by the TAPEINS.CMD indirect command file:

```
*     An error occurred while opening the input file, filename.INS.
*     The FCS or I/O status error code is nn.
*
*     The TAPEINS.CMD procedure did not create a template
*     tape option .INS file for you.
*
*     DD-MMM-YY     HH:MM
```

Explanation: The TAPEINS.CMD procedure cannot open the specified input INS file.

User Action: Refer to the description of <FILERR> in the *Micro/RSX User's Guide, Volume 1* to determine the status of your error code and the necessary corrective action. Repeat the TAPEINS.CMD procedure.

```
*     An error occurred while opening the output file, filename.INS.
*     The FCS or I/O status error code is nn.
*
*     The TAPEINS.CMD procedure did not create a valid template
*     tape option .INS file for you.
*
*     DD-MMM-YY     HH:MM
```

Explanation: The TAPEINS.CMD procedure cannot open your output INS file.

User Action: Refer to the description of <FILERR> in the *Micro/RSX User's Guide, Volume 1* to determine the status of your error code and the necessary corrective action. Delete the invalid template INS file and repeat the TAPEINS.CMD procedure.

```
*     An error occurred while reading from the input file, filename.INS.
*     The FCS or I/O status error code is nn.
*
*     The TAPEINS.CMD procedure did not create a valid template
*     tape option .INS file for you.
*
*     DD-MMM-YY     HH:MM
```

Explanation: The TAPEINS.CMD procedure cannot read from the specified input INS file.

User Action: Refer to the description of <FILERR> in the *Micro/RSX User's Guide, Volume 1* to determine the status of your error code and the necessary corrective action. Delete the invalid template INS file and repeat the TAPEINS.CMD procedure.

```
*     The TAPEINS.CMD procedure did not complete successfully due to
*     the misplacement of at least one FILE or BACKUP__SET statement
*     in the diskette option .INS file.
*
*     Refer to your template tape option .INS file, Tfilename.INS,
*     to identify the misplaced statement(s) and the action you must
*     take before running the TAPEINS.CMD procedure again.
*
*     DD-MMM-YY     HH:MM:SS
```

Explanation: Your input INS file contains a misplaced FILE or BACKUP_SET statement.

User Action: Refer to the invalid template tape option INS file for identification of the misplaced statement. Follow the procedure stated in the template INS file for corrective action. Delete the template INS file and repeat the TAPEINS.CMD procedure.

Appendix A
The MACRO Command

A.1 MACRO

MACRO invokes the MACRO-11 assembler. The MACRO-11 assembler then assembles one or more MACRO-11 source files into a single relocatable object module suitable for processing by the Task Builder.

Format

MACRO[/qualifier...] filespec[/qualifier...] [,filespec,...]

Command Qualifiers

/[NO]CROSS_REFERENCE
/DISABLE:argument
ABSOLUTE
BINARY
CARD_FORMAT
GLOBAL
LOCAL
LOWERCASE
REGISTER_DEFINITIONS
TRUNCATION
/ENABLE:argument
ABSOLUTE
BINARY
CARD_FORMAT
GLOBAL
LOCAL
LOWERCASE
REGISTER_DEFINITIONS
TRUNCATION
/[NO]LIST[:filespec]
/[NO]OBJECT[:filespec]
/[NO]SHOW[:argument]

BINARY
CALLS
COMMENTS
CONDITIONALS
CONTENTS
COUNTER
DEFINITIONS
EXPANSIONS
EXTENSIONS
LISTING_DIRECTIVES
OBJECT_BINARY
SEQUENCE_NUMBERS
SOURCE
SYMBOLS

/[NO]WIDE

File Qualifiers

/LIBRARY
/PASS:n

Parameter

filespec

Specifies input files for the MACRO-11 assembler. These input files must contain MACRO-11 source code. Multiple file specifications must be separated by commas. File specifications must include a file name. If no file type is given, the default file type MAC is applied. If the /LIBRARY file qualifier is used, MLB is the default file type. No wildcards are accepted by MACRO.

Note

The MACRO-11 assembler uses the name of the last file named in the command as the default name for output files. The last file named cannot be a library. All other language commands, and the LINK command, use the first file named in the command as the default name for output files.

Command Qualifiers

/[NO]CROSS_REFERENCE

Specifies whether a cross-reference listing should be generated and appended to the assembly listing. The default is the /NOCROSS_REFERENCE qualifier.

The cross-reference listing locates all user-defined and MACRO symbols that appear in the source program.

When you specify this qualifier, you are also specifying the /LIST qualifier by implication. An assembly listing is generated. It appears in your directory and is also printed on the line printer. If you want to change the name of the listing file, you must use the /LIST qualifier with a file specification in addition to the /CROSS_REFERENCE qualifier.

The /CROSS_REFERENCE qualifier as a command qualifier causes the file to be printed on the line printer; the /CROSS_REFERENCE qualifier as a file specification qualifier prevents the listing file from being printed. See the examples and the discussion under the /LIST qualifier.

/DISABLE:argument
/ENABLE:argument

These qualifiers override the .DSABL and .ENABL assembler directives included in the source program being assembled. The .DSABL and .ENABL directives invoke or inhibit various aspects of the assembly. Table A–1 summarizes the arguments to the /DISABLE and /ENABLE qualifiers and gives their MACRO-11 equivalents. There is a default setting for each of these directives, even if you do not specify them in your program or command line.

These qualifiers affect the entire assembly. If, for example, your MACRO command includes the /ENABLE:LOWER_CASE qualifier, the assembler does not convert any lowercase source text to uppercase, regardless of any .DSABL LC or .ENABL LC directives in the source code. The same applies for the /DISABLE:LOWER_CASE qualifier. All .ENABL LC or .DISABL LC directives are ignored.

If you specify only one argument to the /ENABLE or /DISABLE qualifier, you need not include the parentheses, but if you have more than one argument, they must be separated by commas and enclosed in parentheses.

Table A–1: The Arguments to the /ENABLE and /DISABLE Qualifiers Assembly Functions Disabled by Default

Argument	MACRO-11 Equivalent	Description
ABSOLUTE	AMA	Enabling this function causes relative mode addresses (mode 67) to be assembled as absolute addresses (mode 37).
BINARY	ABS	Enabling this function produces absolute binary output in Files-11 format.
CARD_FORMAT	CDR	Enabling this function causes source columns 73 and greater to be treated as a comment.
GLOBAL	GBL	Disabling this function causes MACRO-11 to treat all symbol references that are undefined at the end of assembly pass 1 as undefined symbols. The default is to treat all such symbols as global symbols.
LOCAL	LSB	Enabling this function permits the disabling or enabling of a local symbol block.
LOWERCASE	LC	Enabling this function causes MACRO-11 to accept lowercase ASCII input. The default is to convert it to uppercase.

Table A–1 (Cont.): The Arguments to the /ENABLE and /DISABLE Qualifiers Assembly Functions Disabled by Default

Argument	MACRO-11 Equivalent	Description
REGISTER_DEFINITIONS	REG	Disabling this function inhibits the normal MACRO-11 default register definitions. The default is R0=%0, R1=%1 . . . SP=%6, PC=%7. Under most circumstances, you should use these defaults.
TRUNCATION	FPT	Enabling this function causes floating-point truncation. The default is floating-point rounding.

/[NO]LIST[:filespec]

Specifies whether an assembly listing should be generated. The default is the /NOLIST qualifier, meaning no assembly listing is generated.

If you do not supply a file specification for this qualifier, the listing has a file name derived from the name of the last source file in the MACRO command with the file type LST. If you want the listing to have a different name, supply the name as an argument to the /LIST qualifier. If you do supply a name, the listing file appears in your directory but is not printed on the line printer.

The /LIST qualifier behaves in a different manner depending on whether it is given as a command qualifier or a file specification qualifier. If the /LIST qualifier is used as a command qualifier, the listing file is placed in your directory and printed on the line printer. If the /LIST qualifier is used as a file specification qualifier, the listing file appears in your directory but is not printed on the line printer.

If your command line includes the /CROSS_REFERENCE qualifier, the /LIST qualifier is implied and need not be specified. The /CROSS_REFERENCE qualifier as a command qualifier causes the file to be printed on the line printer; the /CROSS_REFERENCE qualifier as a file specification qualifier prevents the listing file from being printed.

If your command line includes listing-control arguments to either the /SHOW or /NOSHOW qualifier, the /LIST qualifier is implied and need not be specified. The /[NO]SHOW qualifier as a command qualifier causes the file to be printed on the line printer; the /[NO]SHOW qualifier as a file specification qualifier prevents the listing file from being printed.

The only time you need to use the /LIST qualifier with the /CROSS_REFERENCE or /[NO]SHOW qualifier is when you want to give the listing file a file specification other than the default. In such cases, the position of the qualifiers in the command line has no effect; the listing file appears in your directory but is not printed on the line printer. (See the examples at the end of this section.)

/[NO]OBJECT[:filespec]

Specifies whether an object module should be generated. The default is the /OBJECT qualifier, meaning an object module is generated. If you do not supply a file specification, the object file has a name derived from the name of the last source file and the file type

OBJ. If you want the object file to have a different name, give the name as an argument to the /OBJECT qualifier.

You can name the object file after any of the source files listed in the MACRO command by using the /OBJECT qualifier as a file specification qualifier. If used as a file specification qualifier, the /OBJECT qualifier cannot take a file specification argument.

The /NOOBJECT qualifier specifies that no object module is generated. You can use this qualifier if you want to use other facilities of the assembler, to get an assembly listing, for instance, without doing the assembly.

/[NO]SHOW[:argument]

These qualifiers override the .LIST and .NLIST assembler directives included in the source program being assembled. The .LIST and .NLIST directives control the content and format of the assembly listing. Table A-2 summarizes the arguments to the /SHOW and /NOSHOW qualifiers and gives their MACRO-11 equivalents. There is a default setting for each of these directives, even if you do not specify them in your code or command line.

These qualifiers affect the entire assembly. If, for example, your MACRO command includes the /SHOW:COMMENTS qualifier, the assembly listing includes all comments, regardless of any .NLIST COM or .LIST COM directives in the source code. The same goes for the /NOSHOW:COMMENTS qualifier. All .LIST COM or .DISABL COM directives are ignored.

If you specify only one argument to the /SHOW or /NOSHOW qualifier, you need not include the parentheses, but if you have more than one argument, they must be separated by commas and enclosed in parentheses.

The /SHOW qualifier implies the /LIST qualifier, but if you want the listing file to have a name other than the default, you must still use the /LIST qualifier.

The /SHOW qualifier as a command qualifier causes the file to be printed on the line printer; the /SHOW qualifier as a file specification qualifier prevents the listing file from being printed. See the example and the discussion under the /LIST qualifier.

Table A–2: The Arguments to the /SHOW and /NOSHOW Qualifiers Listing Functions Disabled by Default

Argument	MACRO-11 Equivalent	Description
BINARY	MEB	Enabling this function causes MACRO-11 to list only those macro expansions that generate binary code. This is a subset of EXPANSIONS.
CALLS	MC	Disabling this function suppresses the listing of macro calls and repeat range expansions.
COMMENTS	COM	Disabling this function suppresses the listing of comments. This is a subset of SOURCE.
CONDITIONALS	CND	Disabling this function suppresses the listing of unsatisfied conditional coding.

Table A–2 (Cont.): The Arguments to the /SHOW and /NOSHOW Qualifiers Listing Functions Disabled by Default

Argument	MACRO-11 Equivalent	Description
CONTENTS	TOC	Disabling this function suppresses the listing of table of contents during assembly pass 1. The full assembly listing is still prepared during assembly pass 2.
COUNTER	LOC	Disabling this function suppresses the location counter field and does not replace it with a tab.
DEFINITIONS	MD	Disabling this function suppresses the listing of macro definitions and repeat range expansions.
EXPANSIONS	ME	Enabling this function causes MACRO-11 to include all macro expansions in the listing.
EXTENSIONS	BEX	Disabling this function suppresses the listing of binary extensions, that is, all binary code that will not fit on the first line. This is a subset of OBJECT_BINARY.
LISTING_DIRECTIVES	LD	Enabling this function causes MACRO-11 to list all listing control directives without arguments, that is, those listing directives that alter the listing level count.
OBJECT_BINARY	BIN	Disabling this function suppresses the listing of generated binary code and does not replace it with a tab.
SEQUENCE_NUMBERS	SEQ	Disabling this function suppresses the inclusion of sequence numbers in the listing. Sequence numbers are replaced by tabs.
SOURCE	SRC	Disabling this function suppresses the listing of source lines.
SYMBOLS	SYM	Disabling this function suppresses the listing of the symbol table resulting from the assembly.

/[NO]WIDE

Specifies whether you want the assembly listing in wide or narrow format. The default can be set by your system manager. As supplied, the default is the /WIDE qualifier, also called line printer format. The /NOWIDE qualifier is sometimes called teleprinter format.

This qualifier overrides any .LIST TTM or .NLIST TTM directives included in your source program.

File Qualifiers

/LIBRARY

Specifies that the file is a macro library. The default file type is MLB. A user macro library file must be specified in the command line before any source files that use the macros defined in the library. A library may not be the last file named in the command line. Remember that the system macro library has the file type SML. If you are referencing this library, you must explicitly state the type.

/PASS:n

Specifies that the file is to be assembled only during the pass specified. The assembler makes two passes; n can be either 1 or 2.

Examples

```
$ MACRO [RET]
File? FILEA [RET]
```

Assembles the source file FILEA.MAC into a relocatable object module named FILEA.OBJ.

```
$ MACRO FILEA [RET]
```

Provides the same results as the previous example.

```
$ MACRO FILEA,FILEB,FILEC [RET]
```

Assembles the source files FILEA.MAC, FILEB.MAC, and FILEC.MAC into a relocatable object module named FILEC.OBJ.

```
$ MACRO/LIST FILEA [RET]
```

Assembles the source file FILEA.MAC into the object file FILEA.OBJ, and also produces an assembly listing, FILEA.LST. The assembly listing appears in your directory and is also printed on the line printer.

```
$ MACRO/LIST:MYFILE FILEA [RET]
```

Assembles the source file FILEA.MAC into the object file FILEA.OBJ, and also produces an assembly listing named MYFILE.LST. The assembly listing appears in your directory and is printed on the line printer.

```
$ MAC/LIST:TI:#FILEA [RET]

FILEA  MACRO M1113  08-AUG-86 11:41  PAGE 1

          1                                          .TITLE  FILEA
          2                          .LIST   TTM
          3                          .NLIST  BEX
          4
          5
          6                 ; MACRO LIBRARY CALLS
          7
          8                          .MCALL  EXIT$S,QIOW$,DIR$
          9
               .
               .
               .
```

Assembles the source file FILEA.MAC into the object file FILEA.OBJ and types the assembly listing on your terminal. No permanent copy of the listing file is retained, nor does the listing appear on the line printer.

```
$ MAC/OBJECT:FILED [RET]
```

Produces an object file with the name FILED.OBJ.

```
$ MACRO FILEA/LIBRARY,FILED,FILEF,FILEG [RET]
```

Assembles a concatenated object file named FILEG.OBJ from the source files FILEA.MLB, FILED.MAC, FILEF.MAC, and FILEG.MAC.

```
$ MACRO/NOSHOW:COMMENTS/WIDE FILEA [RET]
```

Produces an object module and an assembly listing. Any .LIST or .NLIST directives with TTM or COM as arguments are ignored. The directives themselves appear in the listing, but they have had no effect. The listing produced by this command includes no comments and is in wide format. The listing is printed on the line printer and appears in your directory as FILEA.LST.

```
$ MACRO/NOSHOW:(COMMENTS, OBJECT_BINARY)/SHOW:EXPANSIONS FILEH [RET]
```

Produces an object module and an assembly listing. Any .LIST or .NLIST directives with COM, BIN, or ME arguments are ignored. The listing includes no comments and no binary code, but it does include macro expansions. Observe the use of parentheses where two arguments are given for the /NOSHOW qualifier and the lack of parentheses where only one argument is given for the /SHOW qualifier.

Appendix B

Prototype Installation Guide for a Diskette Kit

This appendix contains the prototype installation guide for the optional software created in Chapter 8. To produce your installation guide, you must insert into the prototype information specific to your optional software kit. For example, you would replace the symbol <PRODUCT_NAME> with the name of the optional software. If you created a FORTRAN-77 kit, FORTRAN-77 replaces the symbol. Refer to Section 8.3 for more details on the symbols used in this prototype.

B.1 Introduction to Micro/RSX < PRODUCT_NAME> Installation

This guide tells you how to install the Micro/RSX <PRODUCT_NAME> software on your MicroPDP-11 computer. Before you install this software, the Micro/RSX operating system must be installed and working properly. (See the *Micro/RSX Base Kit Installation Guide for Diskettes* for instructions.)

The <PRODUCT_NAME> installation takes only <NUMBER_OF_MINUTES> minutes. Begin by following the instructions in this guide. After you have followed the preliminary instructions in this guide, you follow instructions from your terminal to complete the installation.

A message on your terminal tells you when you have completed the installation.

If you make a mistake or a problem occurs during the installation, see Section B.4.

B.2 Checking Required Software

The Micro/RSX <PRODUCT_NAME> software is distributed on RX50 diskettes. A set of <NUMBER_OF_DISKETTES> diskettes is provided. Check the software box to make sure it contains all <NUMBER_OF_DISKETTES> diskettes.

The set of software consists of the following:

- One diskette labeled <LABEL_NAME>

  <SENTENCE_DESCRIBING_CONTENTS_OF_DISKETTE>

- One diskette labeled <LABEL_NAME>

  <SENTENCE_DESCRIBING_CONTENTS_OF_DISKETTE>

- One diskette labeled <LABEL_NAME>
  <SENTENCE_DESCRIBING_CONTENTS_OF_DISKETTE>
- One diskette labeled <LABEL_NAME>
  <SENTENCE_DESCRIBING_CONTENTS_OF_DISKETTE>

.
.
.

For more information about the <PRODUCT_NAME> software, see the following manuals:

- <MANUAL_TITLE>
- <MANUAL_TITLE>

Remember

Never open a drive door or remove a diskette from its drive, unless both active drive lights are off and both drives are quiet. If a drive door is opened or a diskette removed when the drive lights are on, see Section B.4.

Never turn the power off, unless both active drive lights are off and both drives are quiet. If the power is turned off when the drive lights are on, see Section B.4.

Turn off the power to both the computer and the terminal if you will not be using the system for several hours. The operating system and optional software are saved on the fixed disk when the power is off. You do not have to reinstall them when you turn the power back on.

If you remove a diskette from its drive during the installation procedure—for example, to check that it is inserted properly—you must begin the installation again, going back to step 2 described in Section B.3.

Do not press the RESTART button on the MicroPDP-11 if you make an error.

If the <PRODUCT_NAME> option is already installed on your system, you must remove it before you can install it again.

Replace diskettes in their protective envelopes when not in use.

When the installation is complete, remove the diskette from the drive, replace all diskettes in their protective envelopes, and store them safely.

Store this guide with your diskettes.

B.3 Installing the < PRODUCT_NAME> Software

Before you install the <PRODUCT_NAME> software, make sure the terminal is turned on, the MicroPDP-11 is functional, and the Micro/RSX operating system is installed. See the *Micro/RSX Base Kit Installation Guide for Diskettes* for more information about any of these three items.

If you have more than one terminal, you can use any one for the installation.

Note

If you are using this procedure to replace an option that is already installed, you must remove the installed option first and then install the replacement option. To remove an option, type R and press the RETURN key when the menu appears on your terminal. If you are not sure whether an option is installed, type L and press the RETURN key for a list of installed options.

Convention

The symbol [RET] means that you press the RETURN key.

Follow these steps to install the software:

1. Log in to a privileged account. If you have already done so, do not log in again.

   If you are not sure how to log in, see Section B.5. After you have logged in, proceed to step 2.

2. Insert the diskette labeled <LABEL_NAME> . You can use either drive.

3. Type:

   ```
   $ @OPTION [RET]
   ```

   EXPERIENCED Micro/RSX USERS: You have the option of installing the <PRODUCT_NAME> software on a disk other than the system disk (LB), for example, an optional fixed disk. The installation procedure will ask you for the name of the device you are going to use. To do this, type the following:

   ```
   $ @OPTION /DISK [RET]
   ```

   NOTE TO OPTIONAL SOFTWARE DEVELOPERS: If the user can specify whether he will use the File Control Services (FCS) or Record Management Services (RMS) record-access method, include the following:

   EXPERIENCED <PRODUCT_NAME> USERS: You have the option of conditionalizing <PRODUCT_NAME> during the installation procedure. To do this, simply type:

   ```
   $ @OPTION /FULL [RET]
   ```

   See the <MANUAL_TITLE> for more information on the conditionalization that the /FULL qualifier allows.

   <PARA_DESCRIBING_CONDITIONALS_IF_APPLICABLE>

NOTE TO OPTIONAL SOFTWARE DEVELOPERS: If you are putting more than one software option on a kit, and if any of the options contain any conditionalization, include the following:

Caution

You cannot choose to install ALL the software options on your kit at one time. You must install each software option separately.

Note

If you are not an experienced user, you should use the standard installation procedure.

4. Look at your terminal.

   A menu appears on your terminal. Type I and press the RETURN key to indicate that you want to install software.

   From this point, you follow instructions from your terminal. For example, you type D, for diskette, in response to the question about what type of installation kit you have.

   After <PRODUCT_NAME> is installed, an Installation Verification Procedure (IVP) runs automatically to ensure that the software was installed properly. The IVP ends with the following message:

   ```
   <IVP_MESSAGE>
   ```

   Upon successful completion of the installation procedure, the following message appears on your terminal:

   ```
   Procedure successfully completed.
   ```

Your <PRODUCT_NAME> software option is now fully installed and ready for use. Be sure to remove the diskette from the drive.

The <PRODUCT_NAME> installation procedure copied the following files, (which you need to use this software) onto the fixed disk. Do not delete these files.

Account	File Name	Description
<ACCOUNT>	<FILE>	<DESCRIPTION>
<ACCOUNT>	<FILE>	<DESCRIPTION>
<ACCOUNT>	<FILE>	<DESCRIPTION>
<ACCOUNT>	<FILE>	<DESCRIPTION>
.	.	.
.	.	.
.	.	.

B.4 Correcting Possible Errors

This section provides instructions for correcting errors that may occur during the software installation.

In some cases, the installation software can detect when an error has occurred. In these cases, a message on the terminal provides instructions to help you correct the problem.

Retrying the Installation

To retry the installation, type the following:

```
$ @OPTION [RET]
```

When the menu reappears, type 1 and press the RETURN key to indicate that you want to install software. Make sure that the first or only diskette for the option is in a drive. Do not log in again. Do not press the RESTART button on the MicroPDP-11.

Retrying the Installation Using the Other Drive

Using the other drive and retrying the installation simply means that if the diskette is in the left (top) drive, you should put it in the right (bottom) drive. Or, if the diskette is in the right (bottom) drive, you should put it in the left (top) drive.

Then, to retry the installation, type the following:

```
$ @OPTION [RET]
```

When the menu reappears, type I and press the RETURN key to indicate that you want to install software. Make sure that the first or only diskette for the option is in a drive. Do not log in again. Do not press the RESTART button on the MicroPDP-11.

Correcting Kit, System Disk, and Installation Procedure Errors

During the <PRODUCT_NAME> installation, a message may appear on your screen, indicating that something is wrong with either your <PRODUCT_NAME> software, your system disk, or the installation procedure.

The following subsections describe how to correct many of the kit, system disk, and installation procedure errors.

Correcting Kit Errors

Condition: The wrong diskette is in the drive.
Possible Correction: Locate the correct diskette and, when asked, insert it in the drive.

Condition: You have problems with the drive.
Possible Correction: Try the installation using the other drive. If this doesn't work, look up the error message and its suggested correction in the *Micro/RSX System Manager's Guide.*

Correcting System Disk Errors

Condition: There are undetected bad blocks on the system disk.
Possible Correction: Backup the existing system and then use the ANALYZE/MEDIA command to locate all bad blocks. Using ANALYZE/MEDIA destroys all the data on the disk, so you will have to use the latest backup to restore the disk. If there are no backups of the disk, reinstall the Micro/RSX operating system.

Condition: You have problems with the drive.
Possible Correction: Look up the error message and its suggested correction in the *Micro/RSX System Manager's Guide.*

Condition: There is not enough contiguous or free space on the fixed disk to install the software option.
Possible Correction: You will have to delete files until you have sufficient space to install the option. To find out how much space is available on the fixed disk, type DIR/FREE in response to the dollar sign prompt (**$**).

Condition: Logical device LB is not assigned to the correct device.
Possible Correction: Assign LB to the correct device.

Correcting Installation Procedure Errors

Condition: Status of the system disk or the drive being used to install the <PRODUCT_NAME> software is not as expected.
Possible Correction: Use the SHOW DEVICES command to check that the drive you are using is on line, deallocated, and dismounted and to check that the system drive is mounted public.

Condition: BACKUP utility was aborted.
Possible Correction: Restart the installation procedure.

Correcting Other Possible Errors

Condition: The terminal display does not change.
Possible Correction: Type CTRL/Q by pressing the CTRL key and holding it down while you press Q. If this fails to correct the condition, check to be sure that the terminal is installed properly.

Note

For more information about the terminal and computer setup, see the *MicroPDP-11 System Unpacking and Installation Guide* that came with your hardware.

Condition: Nothing appears on the terminal, or unexpected characters appear on the terminal.
Possible Correction: Ensure that the terminal is plugged in and turned on. Make sure the baud rates on both the MicroPDP-11 computer and the terminal are set to the same speed (typically, 9600). Ensure that the communication cable is firmly connected to the terminal. Turn the terminal off and then back on. Retry the installation beginning with step 1 (logging in).

Condition: Nothing happens when you turn the computer on. (There should be a low whirring noise when the computer is on.)
Possible Correction: Make sure the terminal is plugged in. Make sure the baud rate on the computer is set to the same speed as the terminal (typically, 9600) and that all the hardware is set up and working properly. Retry the installation beginning with step 1 (logging in).

Condition: You opened the door to the drive containing the diskette when the active drive light was on, or you removed the diskette from the drive.
Possible Correction: Reinsert the diskette in the drive, close the drive door, and retry the installation beginning with step 1 (logging in).

Condition: You turned the power off during the installation, or the power failed.
Possible Correction: Remove the diskette from its drive (if a diskette is in one of the drives), turn the power back on, and retry the installation beginning with step 1 (logging in).

Condition: You made an error when typing information at the terminal.
Possible Correction: If you typed an incorrect character, but have not pressed the RETURN key at the end of the line, simply press the DELETE key as many times as necessary to delete the line back to the incorrect character. Then, retype the character correctly and continue. If you have already pressed the RETURN key, you will get an error message. Simply type the information again, making sure that everything is typed correctly. You can always retype a command line or reenter a response to a question.

Note

Do not press the BACKSPACE key to try to delete characters.

Condition: You inserted the diskette incorrectly.
Possible Correction: The installation software can detect this error. When this occurs, a message appears on the terminal instructing you to check to be sure that the diskette is inserted properly. After you check, you can press the RETURN key and continue; or you can remove the diskette, reinsert it properly, and then retry the installation beginning with step 1 (logging in).

Condition: Numerals and the at sign (@) character appear on the terminal.
Possible Correction: An error in the installation procedure has occurred. If a diskette is in one of the drives, remove it from its drive, reboot, and retry the installation beginning with step 1 (logging in).

Condition: One of the following messages appears on your terminal:

```
COP -- Allocation failure -- no contiguous space
COP -- Allocation failure on output file
COP -- Allocation failure -- no space available
```

Possible Correction: Not enough space is available on the fixed disk to install the software option. You will have to delete files until there is sufficient space to install the option. To find out how much space is available on the fixed disk, type DIR/FREE in response to the dollar sign prompt ($).

Condition: The following message appears on your terminal:

```
XXX.INS not found
```

Possible Correction: You do not have the file necessary to complete the installation procedure.

Note

For detailed instructions on properly inserting diskettes in the drives, see the *Micro/RSX Base Kit Installation Guide for Diskettes.*

B.5 Logging in to Micro/RSX

Logging in gives you access to the system. The system identifies the people who have access to it—its users—by means of accounts.

Before you log in, look at your terminal. If the terminal and computer are on, and the operating system is properly installed, you will see a dollar sign prompt. This is the system prompt and indicates that the operating system is ready to accept commands that you type. If you do not see a dollar sign, press the RETURN key; the dollar sign prompt should appear.

The following example of how to log in uses a name and password for a privileged account that is already set up when the Micro/RSX operating system is installed. If your system has been installed for some time, the name and password for this account should have been changed. (See your system manager.)

1. To log in, type LOGIN and press the RETURN key.
2. In response to this command, the system asks for your account or name. Type MICRO and press the RETURN key.
3. Finally, the system asks for your password. Type RSX and press the RETURN key. Note that the password does not appear on your terminal when you type it.

The entire login sequence looks like this:

```
$ LOGIN [RET]
Account or name: MICRO [RET]
Password:
```

The system allows 1 minute for logging in. If you do not complete the login sequence within 1 minute, the following message will appear on your terminal:

```
HEL -- Timeout on response
```

This is nothing to be concerned about. Simply begin the login sequence again by typing LOGIN and pressing the RETURN key.

When you have logged in, you see several messages on the terminal, including the following:

```
Welcome to Micro/RSX
   Version 4.0
```

The messages are followed by the dollar sign prompt. Now you are ready to use the system or to install any optional software you may have purchased.

See the *Introduction to Micro/RSX* for information about the login procedure. See the *Micro/RSX System Manager's Guide* for information about user accounts and managing a Micro/RSX system.

Appendix C
Prototype Installation Guide for a Tape Kit

This appendix contains the prototype installation guide for the optional software created in Chapter 8. To produce your installation guide, you must insert into the prototype information specific to your optional software kit. For example, you would replace the symbol <PRODUCT_NAME> with the name of the optional software. If you created a FORTRAN-77 kit, FORTRAN-77 replaces the symbol. Refer to Section 8.3 for more details on the symbols used in this prototype.

C.1 Introduction to Micro/RSX < PRODUCT_NAME> Installation

This guide tells you how to install the Micro/RSX <PRODUCT_NAME> software on your MicroPDP-11 computer. Before you install this software, the Micro/RSX operating system must be installed and working properly. (See the *Micro/RSX Base Kit Installation Guide for Tape* for instructions.)

The <PRODUCT_NAME> installation takes only <NUMBER_OF_MINUTES> minutes. Begin by following the instructions in this guide. After you have followed the preliminary instructions in this guide, you follow instructions from your terminal to complete the installation.

A message on your terminal tells you when you have completed the installation.

If you make a mistake or a problem occurs during the installation, see Section C.4.

C.2 Checking Required Software

The Micro/RSX <PRODUCT_NAME> software is distributed on a TK50 tape. The box containing the tape is labeled <BOX_LABEL> . The tape itself is labeled as follows:

<LABEL_NAME>

Store the tape in a safe place. Do not modify the contents. For more information about the <PRODUCT_NAME> software, see the following manuals:

- <MANUAL_TITLE>

- <MANUAL_TITLE>

Remember

Turn off the power to both the computer and the terminal if the system will not be used for several hours. The operating system and optional software are saved on the fixed disk when the power is off. You do not have to reinstall them when you turn the power back on.

Do not press the RESTART button on the MicroPDP-11 if you make an error.

If <PRODUCT_NAME> is already installed on your system, you must remove it before you can install it again.

C.3 Installing the < PRODUCT_NAME> SOFTWARE

Before you install the <PRODUCT_NAME> software, make sure the terminal is turned on, the MicroPDP-11 is functional, and the Micro/RSX operating system is installed. See the *Micro/RSX Base Kit Installation Guide for Tape* for more information about any of these three items.

If you have more than one terminal, you can use any one for the installation.

Note

If you are using this procedure to replace an option that is already installed, you must remove the installed option first and then install the replacement option. To remove an option, type R and press the RETURN key when the menu appears on your terminal. If you are not sure whether an option is installed, type L and press the RETURN key for a list of installed options.

Convention

The symbol RET means that you press the the RETURN key.

Follow these steps to install the software:

1. Log in to a privileged account. If you have already done so, do not log in again.

   If you are not sure how to log in, see the section entitled Logging in to Micro/RSX. After you have logged in, proceed to the next step in this section.

2. Insert the tape labeled <LABEL_NAME> .

3. Press the LOAD button. Wait until the light in the LOAD button stops blinking slowly (about 15 seconds).

4. Type:

   ```
   $ @OPTION RET
   ```

   EXPERIENCED Micro/RSX USERS: You have the option of installing the <PRODUCT_NAME> software on a disk other than the system disk (LB), for example, an optional fixed disk. The installation procedure will ask you for the name of the device you are going to use. To do this, type the following:

   ```
   $ @OPTION /DISK RET
   ```

NOTE TO OPTIONAL SOFTWARE DEVELOPERS: If the user can specify whether he will use the File Control Services (FCS) or Record Management Services (RMS) record-access method, include the following:

EXPERIENCED <PRODUCT_NAME> USERS: You have the option of conditionalizing <PRODUCT_NAME> during the installation procedure. To do this, simply type:

```
$ @OPTION /FULL RET
```

See the <MANUAL_TITLE> for more information on the conditionalization that the /FULL qualifier allows.

<PARA_DESCRIBING_CONDITIONALS_IF_APPLICABLE>

NOTE TO OPTIONAL SOFTWARE DEVELOPERS: If you are putting more than one software option on a kit, and if any of the options contain any conditionalization, include the following:

Caution

You cannot choose to install ALL the software options on your kit at one time. You must install each software option separately.

Note

If you are not an experienced user, you should use the standard installation procedure.

5. Look at your terminal.

   A menu appears on your terminal. Type I and press the RETURN key to indicate that you want to install software.

   From this point, you follow instructions from your terminal. For example, you type T, for tape, in response to the question about what type of installation kit you have.

   After <PRODUCT_NAME> is installed, an Installation Verification Procedure (IVP) runs automatically to ensure that the software was installed properly. The IVP ends with the following message:

   ```
   <IVP_MESSAGE>
   ```

   Upon successful completion of the installation procedure, the following message appears on your terminal:

   ```
   Procedure successfully completed.
   ```

Your <PRODUCT_NAME> software option is now fully installed and ready for use. Be sure to unload the tape and remove the tape cartridge from the drive.

The <PRODUCT_NAME> installation procedure copied the following files (which you need to use this software) onto the fixed disk. Do not delete these files.

Account	File Name	Description
<ACCOUNT>	<FILE>	<DESCRIPTION>
<ACCOUNT>	<FILE>	<DESCRIPTION>
<ACCOUNT>	<FILE>	<DESCRIPTION>
<ACCOUNT>	<FILE>	<DESCRIPTION>
.	.	.
.	.	.
.	.	.

C.4 Correcting Possible Errors

This section provides instructions for correcting errors that may occur during the software installation.

In some cases, the installation software can detect when an error has occurred. In these cases, a message on the terminal provides instructions to help you correct the problem.

Retrying the Installation

To retry the installation, type the following:

```
$ @OPTION RET
```

When the menu reappears, type I and press the RETURN key to indicate that you want to install software. Make sure that the tape cartridge is in the drive and that it has been unloaded and reloaded. Do not log in again. Do not press the RESTART button on the MicroPDP-11.

Correcting Kit, System Disk, and Installation Procedure Errors

During the <PRODUCT_NAME> installation, a message may appear on your screen, indicating that something is wrong with either your <PRODUCT_NAME> software, your system disk, or the installation procedure.

The following subsections describe how to correct many of the kit, system disk, and installation procedure errors.

Correcting Kit Errors

Condition: The wrong tape is in the drive.
Possible Correction: Locate the correct tape, dismount the incorrect tape, and mount the correct tape.

Condition: You have problems with the drive.
Possible Correction: Look up the error message and its suggested correction in the *Micro/RSX System Manager's Guide.*

Correcting System Disk Errors

Condition: There are undetected bad blocks on the system disk.
Possible Correction: Backup the existing system, then use the ANALYZE/MEDIA command to locate all bad blocks. Using ANALYZE/MEDIA destroys all the data on the disk, so you will have to use the latest backup to restore the disk. If there are no backups of the disk, reinstall the Micro/RSX operating system.

Condition: There is not enough contiguous or free space on the fixed disk to install the software option.
Possible Correction: You will have to delete files until you have sufficient space to install the option. To find out how much space is available on the fixed disk, type DIR/FREE in response to the dollar sign prompt (**$**).

Condition: Logical device LB is not assigned to the correct device.
Possible Correction: Assign LB to the correct device.

Correcting Installation Procedure Errors

Condition: Status of the system disk or the drive being used to install the <PRODUCT_NAME> software is not as expected.
Possible Correction: Use the SHOW DEVICES command to check that the drive you are using is on line, deallocated, and dismounted, and to check that the system drive is mounted public.

Condition: BACKUP utility was aborted.
Possible Correction: Restart the installation procedure.

Correcting Other Possible Errors

Condition: The terminal display does not change.
Possible Correction: Type CTRL/Q by pressing the CTRL key and holding it down while you type Q. If this fails to correct the condition, check to be sure that the terminal is installed properly.

Note

For more information about the terminal and computer setup, see the *MicroPDP-11 System Unpacking and Installation Guide* that came with your hardware.

Condition: Nothing appears on the terminal, or unexpected characters appear on the terminal.
Possible Correction: Ensure that the terminal is plugged in and turned on. Make sure the baud rates on both the MicroPDP-11 computer and the terminal are set to the same speed (typically, 9600). Ensure that the communication cable is firmly connected to the terminal. Turn the terminal off and then on. Retry the installation as described previously.

Condition: Nothing happens when you turn the computer on. (There should be a low whirring noise when the computer is on.)
Possible Correction: Make sure the terminal is plugged in. Make sure the baud rate on the computer is set to the same speed as the terminal (typically, 9600) and that all the hardware is set up and working properly. Retry the installation beginning with step 1 (logging in).

Condition: You turned the power off during the installation, or the power failed.
Possible Correction: Unload the tape and remove it from the drive (if it is in the drive), turn the power back on, and retry the installation beginning with step 1 (logging in).

Condition: You made an error when typing information at the terminal.
Possible Correction: If you typed an incorrect character, but have not pressed the RETURN key at the end of the line, simply press the DELETE key as many times as necessary to delete the line back to the incorrect character. Then, retype the character correctly and continue. If you have already pressed the RETURN key, you will get an error message. Simply type the information again, making sure that everything is typed correctly. You can always retype a command line or reenter a response to a question.

Note

Do not press the BACKSPACE key to try to delete characters.

Condition: Numerals and the at sign (@) character appear on the terminal.
Possible Correction: An error in the installation procedure has occurred. If the tape is in the drive, unload it, remove it, reboot, and retry the installation beginning with step 1 (logging in).

Condition: One of the following messages appears on your terminal:

```
COP -- Allocation failure -- no contiguous space
COP -- Allocation failure on output file
COP -- Allocation failure -- no space available
```

Possible Correction: Not enough space is available on the fixed disk to install the software option. You will have to delete files until there is sufficient space to install the option. To find out how much space is available on the fixed disk, type DIR/FREE in response to the dollar sign prompt.

Condition: The following message appears on your terminal:

```
XXX.INS not found
```

Possible Correction: You do not have the file necessary to complete the installation procedure.

Note

For detailed instructions on properly inserting a tape cartridge in the drive, see the *Micro/RSX Base Kit Installation Guide for Tape.*

C.5 Logging in to Micro/RSX

Logging in gives you access to the system. The system identifies the people who have access to it—its users—by means of accounts.

Before you log in, look at your terminal. If the terminal and computer are on, and the operating system is properly installed, you will see a dollar sign (**$**). This is the system prompt and indicates that the operating system is ready to accept commands that you type. If you do not see a dollar sign, press the RETURN key; the dollar sign should appear.

The following example of how to log in uses a name and password for a privileged account that is already set up when the Micro/RSX operating system is installed. If your system has been installed for some time, the name and password for this account should have been changed. (See your system manager.)

1. To log in, type LOGIN and press the RETURN key.
2. In response to this command, the system asks for your account or name. Type MICRO and press the RETURN key.
3. Finally, the system asks for your password. Type RSX and press the RETURN key. Note that the password does not appear on your terminal when you type it.

The entire login sequence looks like this:

```
$ LOGIN [RET]
Account or name: MICRO [RET]
Password:
```

The system allows 1 minute for logging in. If you do not complete the login sequence within 1 minute, the following message appears on your terminal:

```
HEL -- Timeout on response
```

This is nothing to be concerned about. Simply begin the login sequence again by typing LOGIN and pressing the RETURN key

When you have logged in, you see several messages on the terminal, including the following:

```
Welcome to Micro/RSX
   Version 4.0
```

The messages are followed by the dollar sign prompt. Now you are ready to use the system or to install any optional software you may have purchased.

See the *Introduction to Micro/RSX* for information about the login procedure. See the *Micro/RSX System Manager's Guide* for information about user accounts and managing a Micro/RSX system.

Appendix D
Micro/RSX Reference and Customization Files

The Advanced Programmer Kit provides the files listed below on the diskette labeled REFERENCE AND CUSTOMIZATION FILES (RX50 distribution) or the backup set labeled REFERENCE (TK50 distribution). These files can be used by a knowledgeable system programmer to customize a Micro/RSX operating system. The files are separated into three major groups. The file specifications and descriptions for each group are as follows:

1. Executive Maps (for reference)

File Name	Description
[1,54]NOIDEXEC.MAP	Map of the Executive for the overlapped instruction and data space Micro/RSX system.
[1,54]IDEXEC.MAP	Map of the Executive for the separated instruction and data space Micro/RSX system.

Note

These two files are usually called RSX11M.MAP. They have been renamed for ease of identification.

2. National Replacement Character set (NRC) terminal support customization

File Name	Description
[12,10]TRANSACD.MAC [12,10]VTMNC.MAC [12,10]VTMNCI.MAC [12,10]VTW.MAC [12,10]VTMNCIG.MAC [12,10]VTMNIF.MAC	Template source files for NRC translation Ancillary Control Drivers (ACDs). Refer to the INSTALL command in the *Micro/RSX User's Guide, Volume 2* for details on modifying these files.
[12,10]ACDBLD.CMD	Task-build command file for NRC translation ACDs.

3. File Control Services (FCS) customization

File Name	Description
[1,1]FCS.OBS	This file contains concatenated FCS object modules. Insert these modules in [1,1]NOANSLIB.OLB (located on the Micro/RSX Base Kit) to obtain an FCS object module with full functionality, including logical name and American National Standards Institute (ANSI) tape support. (DIGITAL recommends that you rename [1,1]NOANSLIB.OLB to [1,1]ANSLIB.OLB, to accurately reflect its contents.) The resulting object library can be modified with the files [1,1]FCSNOLOG.OBS and [1,1]FCSNOANS.OBS to create an FCS object module with other combinations of extended logical name and ANSI tape support.
[1,1]FCSNOLOG.OBS	This file contains concatenated FCS object modules. Replacing the modules contained in [1,1]NOANSLIB.OLB with these modules creates an FCS object module without logical name support. This file can also be combined with FCSNOANS.OBS to create an FCS object module without extended logical name and ANSI tape support.
[1,1]FCSNOANS.OBS	This file contains concatenated FCS object modules. Replacing the modules contained in [1,1]NOANSLIB.OLB with these modules creates an FCS object module without tape support. This file can also be combined with FCSNOLOG.OBS to create an FCS object module without extended logical name and ANSI tape support.

Note

The diskette labeled REFERENCE AND CUSTOMIZATION FILES and the backup set labeled REFERENCE cannot be installed as an application. If you attempt this, the Optional Software Installation procedure returns an error message.

- For RX50 distribution, files can be copied from the diskette by issuing the following commands (in this example the system disk is DU0 and the diskette is in drive DU1):

```
$ MOUNT/NOSHARE DU1: REFERENCE
$ CREATE/DIRECTORY DU0:[directory]
$ COPY/OWN DU1:[directory]file,... DU0:[directory]
```

Parameter

[directory]file,...

Indicates that more than one file and/or directory can be specified.

- For the TK50 distribution, files can be copied by issuing the following commands (in this example the system disk is DU0):

```
$ MOUNT/NOSHARE/FOREIGN MU0:
$ BACKUP/NOINIT/DIR/SAVE_SET:REFERENCE MU0:[directory]file,... DU0:
```

Parameter

[directory]file,...

Indicates that more than one file and/or directory can be specified.

Index

A

Advanced kit
  software options
    file transfer, 1-2
    magnetic tape software support, 1-2
    program development
      advanced MACRO-11, 1-1
      priviledged, 1-2
    terminal emulation, 1-2
/ALL qualifier
  DEASSIGN command, 7-9
  SHOW ASSIGNMENTS command, 7-15
  SHOW LOGICALS command, 7-15
Assembly
  conditional, 1-4
  language, 1-3
    See also MACRO-11
  listing
    examining at a terminal, 3-5
    formatting, 2-7
    generating, 3-4
    page break, 2-7
    printing, 3-6
    spooling, 3-6
    table of contents, 2-7
    terminal format, 2-7
ASSIGN command, 7-3, 7-4, 7-5
  error message, 7-9
  example, 7-7
  qualifiers
    /FINAL, 7-4, 7-5
    /GLOBAL, 7-5
    /GROUP, 7-5
    /LOCAL, 7-5
    /LOGIN, 7-5
    /SYSTEM, 7-5
    /TERMINAL, 7-5
ASSIGN command
  qualifiers (cont'd.)
    /TRANSLATION:FINAL, 7-5
At sign (@)
  ODT, 5-8

B

BACK_UP SET statement
  merging, 8-28
  separating, 8-28
Backslash operator, 5-5
B command
  ODT, 5-6
Breakpoint
  setting in a task, 5-6
Breakpoint register, 5-6

C

Calling module, 6-4
/COMPRESS qualifier
  LIBRARY command, 6-7
Conditional INS File Statements, 8-12
COPY command, 4-2
Create Logical directive, 7-2
/CREATE qualifier
  LIBRARY command, 6-1 to 6-2, 6-3
CRF utility, 1-8
  assembly cross-reference, 3-5 to 3-6
  global cross-reference, 4-4
/CRO qualifier
  See /CROSS_REFERENCE qualifier
/CROSS_REFERENCE qualifier
  LINK command, 1-8, 4-4, 5-2
  MACRO command, 1-8, 3-5
Cross-reference listing
  assembly, 3-5 to 3-6
  global, 4-4

Cross-Reference Processor
  See CRF
CTRL/O command, 3-5
CTRL/Q command, 3-5
CTRL/S command, 3-5
CTRL/U command, 5-3

D

Data block
  local, 2-9
Data storage
  control in assembly language, 1-4
  directive, 1-4
  MACRO-11 definition, 2-9
  program section, 2-9
DCL, 1-3
  commands
    ASSIGN, 7-3, 7-5
    COPY, 4-2
    DEASSIGN, 7-3, 7-9
    DEFINE, 7-1, 7-12
    DIRECTORY, 3-6, 6-4
    LIBRARY, 1-8, 6-1
    LINK, 1-5, 4-1, 5-2, 5-9, 6-4
    MACRO, 3-1, 6-3
    PRINT, 1-11, 3-6
    PURGE, 3-6
    SHOW ASSIGNMENTS, 7-2, 7-15
    SHOW LOGICALS, 7-2, 7-15
    TYPE, 3-5, 4-4
DEASSIGN command, 7-3, 7-9
  error message, 7-11
  example, 7-11
  qualifiers
    /ALL, 7-9
    /GLOBAL, 7-9
    /GROUP, 7-9
    /LOCAL, 7-9
    /LOGIN, 7-9
    /SYSTEM, 7-9
    /TERMINAL, 7-9
Debugging
  MACRO-11 source file, 3-2, 3-3
  ODT, 1-6
  PMD task, 1-7
  $SNAP, 1-7
  task, 4-5, 5-1
  tool
    See ODT
  using map, 5-2, 5-8
/DEBUG qualifier
  LINK command, 5-2
DEC standard editor
  See EDT
Default
  file type
    MACRO-11, 3-4
    TKB, 4-1
  system library search
    MACRO-11, 2-7
    TKB, 1-10, 4-1
  transfer (starting) address, 4-5
DEFINE command, 7-1, 7-3, 7-12
  error message, 7-15
  example, 7-14
  qualifiers
    /FINAL, 7-4, 7-12
    /GLOBAL, 7-12
    /GROUP, 7-12
    /LOCAL, 7-12
    /LOGIN, 7-12
    /SYSTEM, 7-12
    /TERMINAL, 7-12
    /TRANSLATION:FINAL, 7-12
Delete Logical directive, 7-2
Device
  logical, 7-5, 7-9, 7-15
Diagnostic run
  MACRO-11 source file, 3-1 to 3-2
DIGITAL Command Language
  See DCL
Directive
  Create Logical, 7-2
  data storage, 1-4
  Delete Logical, 7-2
  .END, 2-9, 4-2, 4-5
  EXIT$S, 2-7
  general-purpose, 2-7 to 2-8
  .IDENT, 2-6
  .LIST TTM, 2-7
  Macro, 1-4
  .MCALL, 1-9, 2-7, 6-3
  .NLIST, 2-7
  .PAGE, 2-7
  .PSECT, 2-9
  .SBTTL, 2-7
  system, 1-8
  .TITLE, 2-6, 6-4
Directory
  listing, 3-6
  purging, 3-6
DIRECTORY command, 3-6, 6-4

/DISABLE:GLOBAL qualifier
MACRO command, 3-1
Diskette kit
copying
multivolume files, 8-24
single volume files, 8-22
creating, 8-2
INS file, 8-6
INSTALL.DAT file, 8-3
grouping files, 8-3
INS file example, 8-13
INSTALL.DAT file example, 8-5
INSTALL.DAT file statement, 8-4
number of diskettes, 8-3
setting up files, 8-3
Dollar sign ($)
ODT, 5-7, 5-8

E

EDT, 1-3
.END directive, 2-9, 3-3, 4-2, 4-5
Entry point, 6-3
table, 6-3, 6-6
zero entry points, 6-4
EPT
See entry point
Error codes
See Macro-11
Error message
MACRO-11, 3-4
ODT, 5-3
TAPEINS.CMD file, 8-35
TKB, 4-1
TKTN, 4-5
Executive library, 1-10
Executive macro library, 1-8, 1-9
EXELIB.OLB file, 1-10
EXEMC.MLB file, 1-8, 1-9
EXIT$S directive, 2-7

F

FILE.MAC source code, 2-13
Files
copying multivolume, 8-24
copying single volume, 8-22
directory listing, 3-6
listing, 3-5
printing, 3-6
purging, 3-6
source
creating, 2-9
Files
source (cont'd.)
editing, 2-10
File type
LST, 3-4, 6-7
MAC, 3-1
MAP, 4-4
MLB, 6-1
OBJ, 3-4
PMD, 5-9
TSK, 4-1
/FINAL qualifier
ASSIGN command, 7-4, 7-5
DEFINE command, 7-4, 7-12
Format
MACRO-11
source file, 2-1 to 2-4
skeleton, 2-4
statement, 2-1 to 2-4
/FULL qualifier
LIBRARY command, 6-7

G

G command
ODT, 5-6, 5-7
general-purpose directive, 2-7 to 2-8
Global cross-reference
listing, 4-4
Global default
disabling in MACRO-11, 3-1 to 3-2
/GLOBAL qualifier
ASSIGN command, 7-5
DEASSIGN command, 7-9
DEFINE command, 7-12
SHOW ASSIGNMENTS command, 7-15
SHOW LOGICALS command, 7-15
Global symbol, 1-4
entry point, 1-4, 6-3, 6-6
resolution, 4-1, 6-6
undefined
library resolution, 6-5
using library to resolve undefined, 6-6
/GROUP qualifier
ASSIGN command, 7-5
DEASSIGN command, 7-9
DEFINE command, 7-12
SHOW ASSIGNMENTS command, 7-15
SHOW LOGICALS command, 7-15

H

Hardware
  disks, 1-11
  printers, 1-11
  program development, 1-10
  terminals, 1-11

I

.IDENT directive, 2-6
/INCLUDE qualifier
  LINK command, 6-4, 6-5, 6-6
/INSERT qualifier
  LIBRARY command, 6-6
    inserting modules, 6-6
INS file
  creating, 8-6
  example, 8-13
    inserting modules, 8-19
    multiple diskette option, 8-14
    single diskette option, 8-13
    using conditionals, 8-16
    using conditionals and multivolume files, 8-17
  statement, 8-7
    ABORT taskname, 8-7
    BACKUP_SET volname, 8-8
    COMMAND cmd, 8-8
    DELETE filename, 8-8
    ERROR - errmsg, 8-8
    FILE filename /DELETE, 8-8
    FILE filename /KEEP, 8-8
    INSTALL_PROCEDURE filename, 8-9
    INSTALL_VERIFICATION filename, 8-10
    INSTALL filename /COMMON, 8-9
    INSTALL filename /LIBRARY, 8-9
    INSTALL filename /TASK, 8-9
    LIBRARY_DELETE modname1, 8-11
    LIBRARY_INSERT filename, 8-10
    OPTION_VERSION vrsn, 8-11
    REMOVE_PROCEDURE filename, 8-11
    REMOVE taskname, 8-11
    RUN_IMMEDIATE taskname, 8-11
    RUN_SYSTEM taskname, 8-11
    RUN_WAIT taskname, 8-12
    STARTUP_PROCEDURE filename, 8-12
    SYSTEM_VERSION vrsn, 8-12
INS file
  statement (cont'd.)
    ! text, 8-7
    valid conditionals, 8-13
    with conditionals, 8-12
  template, 8-26
    editing, 8-29
    optimizing, 8-27
INSTALL.DAT file
  creating, 8-3
  editing, 8-29
  example, 8-5
  statement, 8-4
    COPY=NO, 8-4
    COPY=YES, 8-4
    DISKETTES=n, 8-4
    OPTION=OPTDESC, [directory]OPTNAME.INS, 8-4
    SYSTEM_VERSION=vrsn, 8-4
Installation
  documenting, 8-34
  producing package, 8-1
  prototype installation guide, 8-34

K

Kit
  packaging, 8-1

L

/LB qualifier
  LINK command, 6-5
Library
  default search of system
    TKB, 4-1
  executive, 1-10
  executive macro, 1-8
  macro, 1-8, 6-1 to 6-2
  maintenance, 6-7
  modules
    inserting, 6-3
  object, 1-9, 6-3
    designating in LINK, 6-6
    designating in TKB, 6-4, 6-5
    resolving undefined global symbols, 6-5
  RMS-11, 1-8, 1-10
  search
    MACRO-11, 2-7
    TKB, 1-10
  squeezing, 6-2
  system, 1-10

Library (cont'd.)
virtual memory management, 1-10
LIBRARY command, 1-8, 6-1
adding a module to a library, 6-6
efficiency, 1-8
inserting modules in a library, 6-6
qualifiers
/COMPRESS, 6-7
/CREATE, 6-1, 6-3
/FULL, 6-7
/INSERT, 6-6
/LIST, 6-6, 6-7
/MACRO, 6-2
/NAMES, 6-7
/OBJECT, 6-3
/REPLACE, 6-6, 6-7
/LIBRARY qualifier
LINK command, 6-5, 6-6
LINE FEED key
closing location
ODT, 5-7
ODT
closing location, 5-4
displaying word on stack, 5-8
opening location, 5-4
opening location
ODT, 5-7
LINK
generating
map
full, 4-4
standard, 4-4
qualifiers
/CROSS_REFERENCE, 5-2
/DEBUG, 5-2
/LONG, 4-4
/MAP, 4-4, 5-2, 5-9
/POSTMORTEM, 5-9
/SYSTEM_LIBRARY_DISPLAY, 4-4
/TASK, 5-2
LINK command, 1-5, 4-1, 5-2, 5-9, 6-4
See also TKB
object library
use, 6-6
PMD task, 5-9
qualifiers
/CROSS_REFERENCE, 1-8, 4-4
/INCLUDE, 6-4, 6-5, 6-6
/LB, 6-5
/LIBRARY, 6-5, 6-6
/MAP, 6-4, 6-6
MAP, 4-4
LINK command
qualifiers (cont'd.)
NOTASK, 4-4
NOWIDE, 4-4
/POSTMORTEM, 5-9
/TASK, 6-4, 6-6
Listing
assembly, 3-4
control, 1-4, 2-7
directory, 3-6
examining at a terminal, 3-5
global cross-reference, 4-4
printing, 3-6
use in debugging, 5-4
/LIST qualifier
LIBRARY command, 6-6, 6-7
MACRO command, 3-1, 3-4
.LIST TTM directive, 2-7
Local data block, 2-9
Local macro definitions, 2-8
/LOCAL qualifier
ASSIGN command, 7-5
DEASSIGN command, 7-9
DEFINE command, 7-12
SHOW ASSIGNMENTS command, 7-15
SHOW LOGICALS command, 7-15
Local symbol definitions, 2-8
Location counter, 1-5
use in debugging, 5-3, 5-4
Logical device, 7-9, 7-15
Logical name
device, 7-5, 7-9
displaying, 7-2, 7-15
format, 7-3, 7-6
introduction, 7-1
table, 7-2
text, 7-12
translation, 7-4
iterative, 7-4
Logical unit number
See LUN
/LOGIN qualifier
ASSIGN command, 7-5
DEASSIGN command, 7-9
DEFINE command, 7-12
SHOW ASSIGNMENTS command, 7-15
SHOW LOGICALS command, 7-15
/LONG qualifier, 4-4
LST file type, 3-4, 6-7
LUN
default by TKB, 4-3

M

Macro
  call
    cross-reference of symbols, 3-5 to 3-6
    resolution, 1-4, 2-7
    unrecognized, 2-7
  directive, 1-4
  library
    adding modules, 6-6
    creating, 6-1
    definitions, 6-3
    listing information, 6-7
    replacing modules, 6-7
    search of system, 1-9, 2-7
  symbol
    definition, 1-9, 2-7, 6-3
MACRO-11
  assembling source file, 3-1 to 3-2
  assembly language, 1-3
  cross-reference listing, 1-5, 3-5 to 3-6
  data storage
    definition, 2-9
  default search of system library, 2-7
  defining local symbols, 2-8
  directives, 1-4
  disabling global default, 3-1
  error code
    A, 3-2
    E, 3-3
    Q, 3-3
    U, 3-3
  error message, 3-1
  listing, 3-4
    generating, 3-4
  location counter, 1-5
  macro
    cross-reference, 3-5 to 3-6
    library usage, 6-3
    symbol, 1-4, 2-7, 6-3
  MACRO command, 1-3, 3-1, 3-4, 6-3
    qualifiers
      /CROSS_REFERENCE, 1-8, 3-5 to 3-6
      /DISABLE:GLOBAL, 3-1
      /LIBRARY, 6-3
      /LIST, 3-1, 3-4, 3-5
      /NOOBJECT, 3-1
      /OBJECT, 3-4
  object module, 3-4
  Programming language, 1-3
  source file, 2-1 to 2-4
MACRO-11
  source file (cont'd.)
    format, 2-1 to 2-4
    skeleton, 2-4
  source input, 1-4
  statement
    format, 2-1 to 2-4
  symbol
    cross-reference, 3-5 to 3-6
    evaluation, 1-4, 3-1, 6-3
  table of contents generation, 2-7
MACRO command
  See MACRO-11
Macro libraries, 1-8
/MACRO qualifier
  LIBRARY command, 6-2
MAC task, 1-3
  See MACRO-11
Magnetic tape software support
  See Advanced kit
Map
  examining at terminal, 4-4
  full, 4-4
  generating, 4-4
  reducing width, 4-4
  stack limits, 5-8
  standard, 4-4
MAP file type, 4-4
/MAP qualifier, 4-4, 5-9
  LINK command, 4-4, 5-2, 6-4, 6-6
.MCALL directive, 1-9, 2-7, 3-3
  using with user macro library, 6-3
Memory allocation file
  See Map
MLB file type, 6-1
Module name, 2-6, 6-3, 6-4
  table, 6-6
    macro library, 6-2
    object library, 6-3, 6-4
Module version, 2-6

N

/NAMES qualifier
  LIBRARY command, 6-7
.NLIST BEX directive, 2-7
NOANSLIB.OLB file, 1-10
/NOOBJECT qualifier
  MACRO command, 3-1
/NOTASK qualifier
  LINK command, 4-4

/NOWIDE qualifier
LINK command, 4-4

O

Object library, 1-9
adding modules, 6-6
creating, 6-3
creating a user, 6-3
default search of system, 1-9, 4-1
DIGITAL-supplied, 1-9
dual use, 6-6
listing information, 6-7
resolving undefined global symbols, 6-5
Object module
concatenated, 4-2 to 4-3
input to TKB, 4-1
MACRO-11, 1-4, 3-4
/OBJECT qualifier, 3-4
LIBRARY command, 6-3
OBJ file type, 3-4
ODT, 1-6, 5-1
at sign (@), 5-8
backslash (\) operator, 5-5
B command, 5-6
breakpoint register, 5-6
changing location contents, 5-7
correcting input, 5-3
dollar sign ($), 5-6, 5-8
error conditions in task, 5-8
examining locations, 5-4, 5-5
forming address, 5-4
G command, 5-6, 5-7
including in a task, 5-1, 5-2
LINE FEED key, 5-5, 5-7, 5-8
P command, 5-7
question mark, 5-3
R command, 5-3
relocation register, 5-2, 5-4
RETURN key, 5-8
setting breakpoints, 5-6
slash (/), 5-4
source listing use, 5-4
SST, 1-6
SST within, 5-8
task control, 1-6
terminating task execution, 5-8
underline prompt, 5-2
X command, 5-8
On-Line Debugging Tool
See ODT
Option
producing package, 8-1
software
See Advanced kit
Optional software files
copying
from diskettes, 8-31
from fixed disk, 8-33
organizing
on the fixed disk, 8-31
using diskettes, 8-29

P

Packaging guidelines, 8-1
.PAGE directive, 2-7
PC
value, 4-5
P command
ODT, 5-7
PDP-11 Processor
address limit, 1-6
PMD file type, 5-9
See also PMD task
PMD task, 1-7, 5-9
enabling with TKB, 5-9
Postmortem Dump
See PMD task
/POSTMORTEM qualifier, 5-9
LINK command, 5-9
PRINT command, 1-11, 3-6
Program
sectioning, 1-4, 2-6, 2-9
user
breakpoints
setting, 5-6
library, 6-1
macro symbol, 6-3
module
name, 2-6
version, 2-6
object library routines, 6-4
overview of development, 1-12 to 1-14
section definiton, 2-9
system subroutines, 1-10
Program counter
See PC
Programming language, 1-3
See also MACRO-11
.PSECT directive, 2-9
PURGE command, 3-6

Q

QMG, 1-11
Question mark symbol (?)
  ODT, 5-3
Queue Manager
  See QMG

R

R command, 5-3
Record Management Services
  See RMS-11
Register
  breakpoint, 5-6
  relocation, 5-2, 5-4
/REPLACE qualifier
  LIBRARY command, 6-6, 6-7
RETURN key
  closing location
    ODT, 5-8
  ODT
    closing location, 5-4
RMS-11 library, 1-8, 1-9, 1-10
RMSLIB.OLB file, 1-10
RMSMAC.MLB file, 1-8, 1-9
RSXMAC.SML file, 1-8, 1-9, 2-7
RUN command, 4-5, 5-2

S

.SBTTL directive, 2-7
SHOW ASSIGNMENTS command, 7-2, 7-15
  example, 7-16
  qualifiers
    /ALL, 7-3, 7-15
    /GLOBAL, 7-3, 7-15
    /GROUP, 7-3, 7-15
    /LOCAL, 7-3, 7-15
    /LOGIN, 7-3, 7-15
    /SYSTEM, 7-3, 7-15
    /TERMINAL, 7-3, 7-15
SHOW LOGICALS command, 7-2, 7-15
  example, 7-16
  qualifiers
    /ALL, 7-15
    /GLOBAL, 7-15
    /GROUP, 7-15
    /LOCAL, 7-15
    /LOGIN, 7-15
    /SYSTEM, 7-15
    /TERMINAL, 7-15
Skeleton source file
  MACRO-11, 2-4
Slash (/) operator
  single
    ODT, 5-4
$SNAP, 1-7, 5-9
  subset of PMD task, 5-9
Snapshot dump
  See $SNAP
Software options
  See Advanced kit
Source file
  creating, 2-12
  file type MAC, 3-1
  MACRO-11
    assembling, 3-1
    error, 3-2, 3-3
    format, 2-1 to 2-4
      skeleton, 2-4
    introduction to, 2-1
    listing, 3-4
    macro library call, 6-3
SST
  ODT, 1-6, 5-8
  PMD task, 1-7
  role in task termination, 4-5
Statement
  MACRO-11, 1-4
    format, 2-1 to 2-4
Symbol
  cross-reference, 3-5 to 3-6
  global
    entry point, 1-4
    resolution, 1-4, 4-1
  local, 1-4, 2-8
    definition, 2-8
    evaluation, 1-4
  macro
    definition, 1-4, 1-9, 2-7, 6-3
  MACRO-11 evaluation, 1-4, 3-1
SYSLIB.OLB file, 1-10
/SYSTEM_LIBRARY_DISPLAY qualifier
  LINK command, 4-4
System directive, 1-8
System library, 1-10
  contributions (in map), 4-4
  macro, 1-8, 1-9
    search, 2-7
  object, 1-10
    search, 4-1

/SYSTEM qualifier
ASSIGN command, 7-5
DEASSIGN command, 7-9
DEFINE command, 7-12
SHOW ASSIGNMENTS command, 7-15
SHOW LOGICALS command, 7-15
System task, 1-1

T

TAPEINS.CMD file
error messages, 8-35
procedure, 8-26
Tape kit
copying files
from diskettes, 8-31
from fixed disk, 8-33
creating, 8-31
INSTALL.DAT file
editing, 8-29
requirements, 8-25
template INS file
creating, 8-26
editing, 8-29
optimizing, 8-27
Task
aborting, 4-5
breakpoints
setting, 5-6
building, 4-1
create, 1-5 to 1-6
CRF, 1-8
debugging, 4-5
MAC, 1-3
macro calls, 6-3
map, 4-3, 4-4
full, 4-4
standard, 4-4
object library routines, 6-4
PMD, 1-7
running, 4-5
system, 1-1
system library contributions, 4-4
termination, 4-5
TKB default conditions, 4-3
transfer (starting) address
default, 4-2, 4-5
defining, 2-9
Task Builder
See TKB
Task image, 1-5, 4-1
creating, 4-1
/TASK qualifier
LINK command, 5-2, 6-4, 6-6
Task termination and notification
See TKTN
Terminal
controlling output, 3-5
examining a listing, 3-5
types, 1-11
/TERMINAL qualifier
ASSIGN command, 7-5
DEASSIGN command, 7-9
DEFINE command, 7-12
SHOW ASSIGNMENTS command, 7-15
SHOW LOGICALS command, 7-15
Text editor
See EDT
Third-party software
packaging, 8-1
.TITLE directive, 2-6, 6-4
TKB, 1-5
default system library, 1-10
error, 4-1, 4-5
functions, 1-5
generating
cross-reference listing, 4-4
map
full, 4-4
standard, 4-4
input, 1-5
LINK command, 1-5, 4-1, 5-2, 5-9, 6-4
object library
designation, 6-3
output, 1-5
qualifier
/CROSS_REFERENCE, 1-8
symbol
undefined, 4-1
task default conditions, 4-3
transfer (starting) address
default, 4-2
TKTN
abort message, 4-5
using with PMD task, 1-7
Transfer (starting) address
defining, 2-9
system treatment of default, 4-2, 4-5
/TRANSLATION:FINAL qualifier
ASSIGN command, 7-5
DEFINE command, 7-12
TSK file type, 4-1
TYPE command, 3-5, 4-4

V

Virtual memory management library, 1-10
VMLIB.OLB file, 1-10

X

X command
  ODT, 5-8

READER'S COMMENTS

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

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.

_____

_____

_____

_____

_____

_____

What kind of user are you? ____ Programmer ____ Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

-- 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

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

READER'S COMMENTS

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

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.

What kind of user are you? ___ Programmer ___ Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

— 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here —