Professional Tool Kit DIBOL User's Guide

Order No. AA-P043C-TK

October 1983

Supersession: This is Version 1.6 and replaces Version 1.5

Operating System: VAX/VMS V3.3
RSX-11M V4.3
RSX-11M-Plus V2.1
P/OS V1.7

Software Version: Professional Developer's Tool Kit V1.7
PRO/Tool Kit V1.0
Professional Host Tool Kit DIBOL V1.6
PRO/Tool Kit DIBOL V1.6

1st Printing, December 1982
First Revision, May 1983
Second Revision, October 1983

CONTENTS

		Page
PREFACE		xi
CHAPTER 1	INTRODUCTION	
1.1	THE DIBOL TOOL KIT	1-1
1.1.1	The DIBOL Tool Kit Software	1-1
1.1.2	Documentation	1-2
1.2	HOST ENVIRONMENT	1-2
1.3	PROFESSIONAL SYSTEM ENVIRONMENT	1-3
1.4	TOOL KIT DIBOL	1-4
CHAPTER 2	DIBOL PROGRAM DEVELOPMENT FOR THE PROFESSIONAL	
2.1	APPLICATION DEVELOPMENT OVERVIEW	2-2
2.2	PROFESSIONAL SYSTEM CONSIDERATIONS FOR DIBOL PROGRAMMERS	2-4
2.2.1	Interface to the User	2-4
2.2.2	Display	2-4
2.2.2.1	Terminal Addressing	2-4
2.2.2.2	Display Characters	2-4
2.2.3	Keyboard	2-5
2.2.4	P/OS Services and Facilities	2-5
2.2.5	Libraries	2-7
2.2.6	Files	2-7
2.2.7	DIBOL Debugging Technique (DDT)	2-8
2.2.8	Error Messages (Condition Handling)	2-8
2.3	DEVELOPMENT TASKS/STEPS	2-8
2.3.1	Coding/Editing	2-10
2.3.2	Compiling	2-10
2.3.3	Application Building	2-10
2.3.4	Application Installation File	2-10
2.3.5	Transfer to the Professional System	2-10
2.3.6	Installing/Executing/Debugging	2-11
2.3.7	Creating Usable Diskettes	2-11
2.4	PROGRAM MIGRATION	2-11
CHAPTER 3	THE DIBOL COMPILER	
3.1	CHARACTERISTICS	3-1
3.1.1	Compiler Features	3-1
3.2	USING THE DIBOL COMPILER	3-2
3.2.1	Running the DIBOL Compiler	3-2
3.2.2	Command Qualifiers	3-4
3.2.3	Specifying Output Files	3-6

		Page
3.2.4	Contents of the Listing File	3-7
3.2.4.1	The Program Listing	3-7
3.2.4.2	The Symbol Table	3-7
3.2.4.3	The Label Table	3-7
3.2.5	Error Messages	3-8
3.2.6	Example DIBOL Program and Program Listing	3-9
CHAPTER 4	**PROFESSIONAL APPLICATION BUILDER (PAB)**	
4.1	LIBRARIES	4-2
4.2	APPLICATION-BUILD PROCEDURE DECISIONS	4-3
4.2.1	Generating the Command and Descriptor Files	4-3
4.2.2	Editing the Command (.CMD) File	4-4
4.2.3	Editing the Overlay Descriptor File	4-7
4.3	APPLICATION-BUILDING DIBOL PROGRAMS	4-7
4.3.1	A Simple Application Build	4-7
4.3.2	Background Information	4-7
4.3.3	Application Builder Error Messages	4-8
4.3.4	Command Format	4-9
4.3.5	Files Used by the Application Builder	4-11
4.4	EXAMPLES OF .ODL FILES	4-12
4.4.1	Without Overlays	4-12
4.4.2	With Overlays	4-13
CHAPTER 5	**RMS-11 AND DIBOL ON THE PROFESSIONAL**	
5.1	RMS-11 FILE ORGANIZATION	5-1
5.1.1	Sequential File Organization	5-2
5.1.2	Relative File Organization	5-2
5.1.3	Indexed File Organization	5-2
5.1.3.1	Indexed File Keys	5-3
5.1.3.2	Defining Keys	5-4
5.2	RMS-11 ACCESS MODES	5-5
5.2.1	Sequential Access Mode	5-5
5.2.2	Random Access Mode	5-6
5.3	CREATING AN RMS-11 FILE	5-6
5.3.1	The Formats, Lengths, and Storage of Records	5-6
5.3.1.1	Record Formats	5-6
5.3.1.2	Record Lengths	5-7
5.3.1.3	Record Storage	5-8
5.3.2	RMS-11 File Size	5-9
5.4	USING RMS-11 FILES	5-9
5.4.1	Sequential File Record Operations	5-9
5.4.2	Relative File Record Operations	5-9
5.4.3	Indexed File Record Operations	5-10
5.4.4	Block Mode Access	5-10
5.4.5	Record Operations and RMS-11 MACROS	5-10

		Page
5.5	RMS-11 ERROR MESSAGES	5-12
5.6	AN EXAMPLE: CREATING AN INDEXED FILE	5-12
5.7	DATA STORAGE SPACE REQUIREMENTS	5-17
CHAPTER 6	THE DIBOL DEBUGGING UTILITY (DDT)	
6.1	FEATURES	6-1
6.2	PREPARING FOR DDT	6-1
6.2.1	Compiling	6-1
6.2.2	Application Building	6-1
6.2.3	DDT Operation	6-2
6.2.3.1	Running DDT	6-2
6.2.3.2	Using a Terminal Connected to the Printer Port	6-2
6.2.3.3	Failure to Properly Prepare for DDT	6-2
6.2.3.4	Error Messages	6-2
6.3	DDT COMMANDS	6-2
6.3.1	Program Execution Control	6-3
6.3.1.1	Program Execution	6-3
6.3.1.2	Single Step	6-3
6.3.2	Breakpoint Control	6-4
6.3.2.1	Setting Breakpoints	6-5
6.3.2.2	Clearing Breakpoints	6-5
6.3.2.3	Iteration of Breakpoints	6-6
6.3.3	Variable Manipulation	6-7
6.3.3.1	Setting Variables	6-7
6.3.3.2	Examining Variables	6-8
6.3.3.3	Extended Variable Manipulation	6-8
6.3.4	Subroutine Traceback	6-9
CHAPTER 7	DIBOL INTERFACE TO FMS	
7.1	DIBOL DATA TYPES FOR FORM DRIVER ARGUMENTS	7-1
7.2	SYNTAX FOR THE CALLS	7-2
7.3	BUILDING A DIBOL TASK	7-5
CHAPTER 8	DIBOL INTERFACE TO P/OS SYSTEM SERVICES, CALLABLE IMAGES, AND ROUTINES	
8.1	MENU SERVICE ROUTINES	8-2
8.1.1	Open Menu File	8-2
8.1.2	Read Menu Frame	8-2
8.1.3	Show Single-Choice Menu	8-3
8.1.4	Unpack Menu Buffer	8-4
8.1.5	Pack Dynamic Single-Choice Menu	8-6
8.1.6	Display Dynamic Menu	8-7
8.1.7	Pack Multiple-Choice Menu	8-8

		Page
8.1.8	Display Multiple-Choice Menu Frame	8-9
8.1.9	Close Menu File	8-11
8.2	HELP SERVICE ROUTINES	8-12
8.2.1	Open Help File	8-12
8.2.2	Specify Help Frame	8-12
8.2.3	Display Help Frame	8-13
8.2.4	Close Help File	8-14
8.3	MESSAGE SERVICE ROUTINE	8-14
8.4	MISCELLANEOUS SERVICES	8-15
8.4.1	Fatal Error	8-15
8.4.2	Get Keystroke	8-16
8.4.3	Parse String	8-16
8.4.4	New File	8-17
8.4.5	Old File	8-18
8.4.6	Wait for Resume Key	8-19
8.4.7	Send Message to Message/Status Display	8-20
8.5	CALLABLE P/OS USER SYSTEM SERVICES	8-21
8.5.1	PROSE Text Editor	8-21
8.5.2	PRINT Services	8-22
8.5.3	PRO/SORT	8-23
8.6	CALLABLE SYSTEM ROUTINES	8-23
8.6.1	PRODIR	8-24
8.6.2	PROFBI	8-25
8.6.3	PROLOG	8-25
8.6.4	PROVOL	8-26
CHAPTER 9	DIBOL INTERFACE TO THE CORE GRAPHICS LIBRARY	
9.1	CHAPTER ORGANIZATION	9-1
9.2	THE DIBOL INTERFACE	9-1
9.2.1	Calling CORE Graphics Library Subprograms	9-1
9.2.2	Building Your DIBOL Program	9-2
9.2.3	Running CORE GRAPHICS Library Programs	9-4
9.3	CONTROL INSTRUCTIONS	9-4
9.3.1	INITIALIZE_CORE -- Prepare Graphics System for Use	9-4
9.3.2	TERMINATE_CORE -- Graphics System Usage Finished	9-5
9.3.3	REPORT_MOST_RECENT_ERROR -- Identify Execution Error	9-5
9.3.4	NEW_FRAME -- Refresh Screen	9-5
9.3.5	ERASE VIEWPORT -- Erase Images in Viewport	9-6
9.3.6	PRINT_SCREEN -- Send Screen Image to Output Device	9-6
9.3.7	CGL_WAIT -- Suspend Execution	9-7

Contents (Cont.)

		Page
9.4	VIEWING TRANSFORMATION INSTRUCTIONS	9-7
9.4.1	SET_WINDOW - Specify Visible Part of World Coordinate Space	9-7
9.4.2	SET_ORIGIN -- Specify Origin of Window	9-7
9.4.3	SET_WINDOW_CLIPPING -- Enable or Disable Window Clipping	9-8
9.4.4	SET_VIEWPORT_2 -- Specify Usable Area of Screen	9-8
9.4.5	SCROLL -- Move Screen Contents	9-10
9.5	GLOBAL ATTRIBUTE INSTRUCTIONS	9-10
9.5.1	SET_WRITING_INDEX -- Select Color Map Index for Images	9-10
9.5.2	SET_BACKGROUND_INDEX -- Set Background Color Map Index	9-10
9.5.3	SET_COLOR_MAP_ENTRY -- Set Color Map Entry RGB Values	9-11
9.5.4	SET_COLOR_MAP -- Set All Color Map RGB Values	9-12
9.5.5	SET_WRITING_PLANES -- Select Combination of Planes	9-12
9.5.6	SET_WRITING_MODE -- Set Writing Characteristics	9-13
9.5.7	SET_GLOBAL_ATTRIBUTES -- Set Global Attribute List	9-13
9.6	CURRENT POSITION AND MARKER INSTRUCTIONS	9-14
9.6.1	Current Position Instructions	9-14
9.6.1.1	MOVE_ABSOLUTE_2 -- Move to Absolute Position	9-14
9.6.1.2	MOVE_REL_2 - Move to Relative Position	9-14
9.6.1.3	INQUIRE_CURRENT_POSITION_2 -- Get Current Position	9-15
9.6.2	Marker Primitive Instructions	9-15
9.6.2.1	MARKER_ABS_2 -- Draw Marker at Absolute Position	9-15
9.6.2.2	MARKER_REL_2 -- Draw Marker Relative to Current Position	9-15
9.6.2.3	POLYMARKER_ABS_2 -- Draw Markers at Absolute Positions	9-16
9.6.2.4	POLYMARKER_REL_2 -- Draw Markers at Relative Positions	9-16
9.6.3	Marker Attribute Instuctions	9-17
9.6.3.1	SET_MARKER_SYMBOL -- Select New Marker Symbol	9-17
9.7	LINE INSTRUCTIONS	9-18
9.7.1	Straight Line Primitive Instructions	9-18
9.7.1.1	LINE_ABS_2 -- Draw Line to Absolute Position	9-18

Contents (Cont.)

		Page
9.7.1.2	LINE_REL_2 -- Draw Line to Relative Position	9-18
9.7.1.3	POLYLINE_ABS_2 -- Draw Lines to Absolute Positions	9-18
9.7.1.4	POLYLINE_REL_2 -- Draw Lines to Relative Positions	9-19
9.7.1.5	POLYGON_ABS_2 -- Draw Polygon by Absolute Positions	9-19
9.7.1.6	POLYGON_REL_2 -- Draw Polygon by Relative Positions	9-20
9.7.1.7	RECTANGLE_ABS_2 -- Draw Rectangle by Absolute Position	9-20
9.7.1.8	RECTANGLE_REL_2 -- Draw Rectangle by Relative Position	9-21
9.7.2	Curved Line Primitive Instructions	9-21
9.7.2.1	ARC_ABS_2 -- Draw Arc Based on Absolute Position	9-21
9.7.2.2	ARC_REL_2 -- Draw Arc Based on Relative Position	9-21
9.7.2.3	CURVE_ABS_2 -- Draw Curve by Absolute Position	9-22
9.7.2.4	CURVE_REL_2 -- Draw Curve by Relative Position	9-22
9.7.3	Line Attribute Instructions	9-23
9.7.3.1	SET_LINESTYLE -- Set Line Drawing Style	9-23
9.7.3.2	SET_LINEWIDTH -- Set Line Drawing Width	9-23
9.7.3.3	SET_FILL_MODE -- Enable or Disable Area Fill	9-24
9.7.3.4	SET_FILL_ENTITY -- Set Fill to Line or Point	9-24
9.7.3.5	SET_FILL_CHAR -- Specify Character for Fill	9-25
9.8	TEXT INSTRUCTIONS	9-25
9.8.1	Text Primitive Instructions	9-25
9.8.1.1	TEXT -- Draw Line of Text	9-25
9.8.1.2	INQUIRE_TEXT_EXTENT_2 -- Report Position at End of String	9-26
9.8.2	Text Attribute Instructions	9-26
9.8.2.1	SET_CHARSIZE -- Set Character Size	9-26
9.8.2.2	SET_CHARSPACE -- Set Character Spacing	9-27
9.8.2.3	SET_CHARPATH -- Set Text Writing Direction	9-27
9.8.2.4	SET_CHARJUST -- Set Text Justification	9-28
9.8.2.5	SET_CHARITALIC -- Set Character Slant	9-28
9.8.2.6	SET_FONT -- Set Character Font	9-29
9.8.2.7	SET_FONT_SIZE -- Define Size of Character Font	9-29

		Page
9.8.2.8	LOAD_CHARACTER -- Load User-defined Character	9-30
9.9	CONSTANT DECLARATION FILE	9-30
9.10	EXAMPLES	9-33
APPENDIX A	Character Collating Sequence	A-1
APPENDIX B	Multi-National Characters	B-1
APPENDIX C	DIBOL OSSL EXTERNAL SUBROUTINES	C-1
C.1	FUNLK	C-2
C.2	ISMCRE	C-3
C.3	PRINT	C-6
C.4	R5ASC	C-8
C.5	SYSID	C-9
C.6	TBBIN	C-10
C.7	TBDUT	C-11
C.8	TBENG	C-12
C.9	TBFRE	C-13
C.10	TBGER	C-14
C.11	TBMOD	C-15
C.12	TBMUL	C-16
C.13	TBRET	C-17
C.14	TBRPL	C-18

FIGURES

2-1	Application Program Development Steps	2-2
2-2	DIBOL Application Program Development Tasks	2-9
5-1	Single-Key Indexed File Organization	5-3
5-2	MultiKey Indexed File Organization	5-4

TABLES

3-1	Contents of the Symbol Table	3-8
3-2	Contents of the Label Table	3-8
5-1	Permissible Combinations of File Organizations and Record Formats	5-7
5-2	RMS-11 Record Operations	5-11
7-1	Argument Data Types	7-1
7-2	DIBOL Form Driver CALLS	7-2
A-1	English, German, Dutch, and French Collating Sequences	A-2
A-2	Multi-National Collating Sequence	A-3
B-1	ASCII Character Set	B-2
B-2	Mult-National Character Set	B-3

INDEX INDEX-1

PREFACE

This manual provides information on the use of **Professional Host Tool Kit DIBOL** or **PRO/Tool Kit DIBOL** in the development and production of DIBOL programs that are run on a Professional 300 system.

A DIGITAL Professional 300 system executes a DIBOL program that is developed on a host computer system or on the Professional 300 system itself.

The host system may have any of the following operating systems: VAX/VMS V3.3 or later, RSX-11M V4.3 or later, RSX-11M-PLUS V2.1 or later. The **Professional Host Tool Kit DIBOL V1.6** package is used to develop the DIBOL programs on the host and to transfer the developed programs to the Professional 300 system. This process is described in the **Professional 300 series Tool Kit User's Guide.**

The Professional 300 system has the P/OS V1.7 operating system. The **PRO/Tool Kit DIBOL V1.6** package is used to develop the DIBOL programs on the Professional 300 system.

The user of this manual must be a DIBOL programmer who has a working knowledge of the Professional 300 system and, if a host system is used, the operating system of that host (VAX/VMS, RSX-11M-PLUS, or RSX-11M).

RELATIONSHIP TO OTHER DOCUMENTATION

This user's guide provides information specific to DIBOL program development and DIBOL program migration. Use of the Professional Tool Kit components is described in the Professional documentation (see the list of related documentation below). Operations discussed in the Professional documentation which are essential to the DIBOL program developer include: communicating with the host from a Professional system; using the Professional Host Tool Kit on the host; using the PRO/Tool Kit on the Professionl system; transferring programs from the host to the Professional system; and using the FMS interface, the P/OS system services, and the interface to CORE Graphics. Frequent reference is made to the Professional documentation.

STRUCTURE

Individual chapters are devoted to background information, to DIBOL program development for the Professional in general, and to the particular tasks required to produce a program with the DIBOL Tool Kit. These chapters are followed by three appendixes.

Chapter 1	contains background information on the relationship between DIBOL and the Professional system. Environmental factors impacting development and operation are discussed in terms of both the Professional system and DIBOL.
Chapter 2	provides an overview of the required activities for DIBOL program development. The characteristics of the Professional system that impact DIBOL are discussed in detail. Files, utilities, etc., are identified.
Chapter 3	describes how to use the DIBOL compiler.
Chapter 4	explains task building (application building) for DIBOL programs.
Chapter 5	contains information about RMS files that is of particular interest to the DIBOL programmer.
Chapter 6	describes how to use DDT (the DIBOL Debugging Technique) on the Professional system.
Chapter 7	describes the DIBOL interface to FMS.
Chapter 8	describes the DIBOL interface to P/OS system services.
Chapter 9	describes the DIBOL interface to the CORE Graphics Library.
Appendix A	contains two tables illustrating character collating sequences.
Appendix B	contains tables illustrating the ASCII and the multi-national character sets.
Appendix C	describes the DIBOL external subroutines that are used solely with Professional Tool Kit DIBOL.

Document Symbols

The symbols defined below are used throughout this manual:

Symbol	Definition
afield	The name of an alpha field
aliteral	An alpha literal which must be delimited by single quotes; that is, 'ABC'

ch	A decimal expression or symbol that evaluates to an input/output channel number (range: 1-15)
dexp	A decimal expression
dfield	The name of a decimal field
dliteral	A decimal literal
field	The name of a field
filespec	The name of a record, an alpha field, or an alpha literal that contains a file specification in the following form: dev:[directory]filnam.ext;n where: dev: The mnemonic name of an I/O device directory A one to nine character directory name filnam A one to nine character file name; the first character must be alphabetic .ext A one to three character file name extension ;n The version number
label	A statement label
lowercase	Lowercase characters indicate elements of the language that are supplied by the programmer.
nontrappable error	An error that causes program termination and cannot be trapped
record	The name of a record
subnam	The name of a subroutine
trappable error	An error that can cause program termination but may be trapped using the ONERROR statement
UPPERCASE CHARACTERS	Uppercase characters indicate elements of the language that must be used exactly as shown.
<CR>	Line terminator for terminal input; causes CR (carriage Return) and LF (Line Feed) code combination to be sent to the receiving program and echoed at the terminal

[ ]	Indicates optional arguments
| |	Verticle lines indicate that a single choice must be made from a list of arguments.
...	Indicates optional continuation of an argument list in the form of the last specified argument
. . .	Indicates that the example of a program, user input, or system output is sufficient for illustration, but not complete

DOCUMENTATION CONVENTIONS

The following are the documentation conventions used in this manual:

<prompt>	is used to indicate the host prompt character(s).
<CR>	is used to indicate a carriage return used as a line terminator or response during an interactive session.

RELATED DOCUMENTATION

Professional Installation Instructions

Professional Owner's Manual

Professional for Beginners: Hard Disk System

Professional User's Guide for Hard Disk System

Professional User's Guide for Diskette System

Professional Reference Card Diskette System

Professional System Reference Card

PRO/BASIC Language Manual

Professional Communications Manual

Documentation Guidelines

Tool Kit Documentation Directory

Tool Kit Installation Guide and Release Notes

Tool Kit User's Guide

CORE Graphics Library Manual

P/OS System Reference Manual

Terminal Subsystem Manual

IAS/RSX-11 ODT Supplement

IAS/RSX-11 ODT Reference Manual

Host Communications Installation Procedures

PRO/RMS-11: An Introduction

PRO/RMS-11 MACRO Programmers Guide

FMS-11/RSX-11 Software Reference Manual

FMS-11/RSX-11 Release Notes

PRO/FMS-11 Documentation Supplement

RSX-11M/M-PLUS Task Builder Manual

PDP-11 MACRO-11 Language Reference Manual

Introduction to DIBOL

Professional Tool Kit DIBOL Release Notes and Installation Guide

Professional Tool Kit DIBOL Message Manual

DIBOL-83 Language Reference Manual

DIBOL-83 Compatability Guide

CHAPTER 1
INTRODUCTION

Professional 300 system application programs can be developed on either the Professional 300 system itself or on a host computer system; the operating system on the host is VAX/VMS V3.3, RSX-11M V4.3 or RSX-11M-PLUS V2.1 or later.

Application program development on both the Professional 300 and the host are facilitated by software packages called tool kits. The **Professional Host Tool Kit** package is used to develop programs on a host for later execution on Professional 300 system. The **PRO/Tool Kit** package is used to develop programs on the Professional 300 for later execution on that system. These packages are also called Professional tool kit(s).

The Professional tool kits are used together with language tool kits to develop programs in a particular language. DIBOL program development uses the **Professional Host Tool Kit DIBOL** package on a host computer system, and the **PRO/Tool Kit DIBOL** package on the Professional 300 system. These packages are also called the DIBOL tool kit(s).

The following sections describe the host computer system environment, the Professional 300 environment, the components of the DIBOL tool kits, and the associated DIBOL documentation. This information pertains to the development of new DIBOL applications programs and to the migration of existing DIBOL applications.

1.1 THE DIBOL TOOL KIT

1.1.1 The DIBOL Tool Kit Software

Both the **Professional Host Tool Kit DIBOL** package and the **PRO/Tool Kit DIBOL** package consist of the following software components.

- The DIBOL compiler (DIB83P.EXE on VAX, and DIB83P.TSK on RSX or P/OS).
- The DIBOL OSSL and UESL libraries (DBPOSL.OLB AND DBPLIB.OLB).
- The DDT object module (DBLDDT.OBJ).
- The run-time system modules for task linkage (DBLRES.STB and DBLRES.TSK).

This software is described in Chapters 2 and 4 of this manual.

1.1.2 Documentation

The documentation associated with the DIBOL tool kits consists of three manuals.

- The Professional Tool Kit DIBOL User's Guide
- The Professional Tool Kit DIBOL Message Manual
- The Professional Tool Kit DIBOL Release Notes and Installation Guide

The User's Guide provides the information necessary to the following.

- Program development on either the host system or the Professional 300 system.
- Program debugging and execution on the Profesional 300 system.

The **Professional Tool Kit DIBOL Message Manual** describes the messages that are generated by the various components of the DIBOL tool kits. The messages report current status, successful completion, and error detection.

The **Professional Tool Kit DIBOL Release Notes and Installation Guide** describes the features of the latest release of the DIBOL tool kits and the procedure by which the tool kits are installed on the appropriate system.

The software facilities provided by the the Professional tool kits is described in the Professional 300 Series documentation. This Professional tool kit documentation refers to program development in all supported languages (DIBOL, BASIC, PASCAL...). In this manual, reference is made to Professional tool kit documentation as it applies to DIBOL program development.

1.2 HOST ENVIRONMENT

The programmer who is planning DIBOL program development on a host system must be familiar with the host operating system environment (RSX-11M, RSX-11M-PLUS or VAX/VMS with AME). You must be familiar enough with the host system command language (DCL or MCR) to log in, to manipulate files, and to run programs such as the compiler and the Professional Application Builder (PAB).

The host is described from the Professional system perspective in the Professional system documentation and in host documentation. The use of the DIBOL tool kits is described in this manual.

Typically, the Professional system is used (with the communications facility) in the terminal emulation mode operating as a remote work station to perform development tasks on the host. In this mode the operation is the same as any other work station (VT100/VT125 terminal); in fact, development can be accomplished using one of these other "standard" terminals connected directly to the host. The use of the Professional system is necessary, however, for 8-bit characters (see the **Terminal Subsystem Manual**). The Professional system is also used to transfer the executable files (and any other required files) to the Professional system from the host.

The software on the host that is supplied with the DIBOL Tool Kit consists of DIBOL object libraries and DIBOL run-time system modules that are used for symbol reference resolution during application building on the host. Other DIBOL Tool Kit software on the host includes the DIBOL compiler and the DDT object module.

1.3 PROFESSIONAL SYSTEM ENVIRONMENT

Both programmers who develop their programs on another host and programmers who develop their programs on the PRO itself must be familiarwith the PROFESSIONALIS SYSTEM ENVIRONMENT; DIBOL programs are installed and executed in this environment.

The Professional 300 system environment is a single user multitask operating system, P/OS. Depending on the hardware configuration, P/OS is either disk or diskette based. P/OS features include a standard interface to the end user via menus that can be integrated into an application. Other features include access to user-developed help frames and application messages. File support is provided by RMS (Record Management Services).

The Professional 300 system supports clustered and segmented resident libraries that are used for RMS file I/O, for P/OS system functions and services, and for the DIBOL run-time interpreter.

The Professional system does not require traditional device codes. Internally the "dev:[directory]filename.ext" format is accepted; at the user level this is optional and all the system devices are accessed by name. For example, when a diskette is inserted into a drive it is automatically mounted and causes that drive to assume the name (volume label) assigned to the diskette when it was created with the application builder program.

When the application is installed on a Professional system, it is done by means of an "application installation file". This file contains information related to the application configuration and the software component parts of the total application including a degree of control over device names. The application installation utility is selected from the disk service menu.

The P/OS operating system "command interface" is called the ProDispatcher. The ProDispatcher (via the installation file) starts and terminates applications. The ProDispatcher also controls the main menu and the user interface.

The P/OS operating system provides an intertask communications mechanism to the ProDispatcher. It provides access to the terminal driver (and, through this driver, to the printer), to device drivers, and makes the installed diskette name an available volume. It also supports the various RMS and P/OS file, print, and other system services.

1.4 TOOL KIT DIBOL

Tool Kit DIBOL application program support consists of a shared/-resident library for the run-time interpreter, RMS, and P/OS. Subroutines may be linked to the task from object libraries on the host. DIBOL on the Professional is designed to be highly compatible with DIBOL on CTS-300, RSTS/E, and VAX-11. Some differences between DIBOL on the Professional and the other DIBOL versions do exist. The differences are due to operating system functionality and file management system differences (i.e., RMS versus DMS -- Data Management Services -- file management). Those procedure statements and external subroutines with differences are listed and explained in the DIBOL-83 Compatibility Guide.

DIBOL on the Professional allows users to take advantage of many Professional system features, because the DIBOL Tool Kit has (as part of the OSSL library) interface routines to the Professional system services. More information on DIBOL and access to the system services and features is described in Chapters 7, 8, and 9 of this manual.

CHAPTER 2

DIBOL PROGRAM DEVELOPMENT FOR THE PROFESSIONAL

The process of DIBOL program development can take place in a host environment or in a Professional 300 system environment. These environments and their associated software packages are described in Chapter 1.

If the host environment is used, the Professional is used as a work station (remote terminal connected to the host) for program development; i.e., source program creation, compilation, debugging and linking. After compiling and linking, the executable program is transferred from the host to the Professional system; and final debugging of the program is done on the Professional.

If the Professional 300 system environment is used, the Professional provides the sole environment for all programming activities; i.e., the Professional provides the environment for program development, compilation, debugging, and execution.

The documentation required for program development in these two environments is described in Chapter 1.

NOTE

Effective with Version 1.5, the following items should be particularly noted:

- There are important differences when creating applications that will run under P/OS Diskette. These differences are covered in the **Professional Tool Kit User's Guide.** This user's guide focuses on development of applications for P/OS hard disk.

- There is a new resident run-time library, DBLRES, that programs are built against on the host and which supports new applications on the Professional. The previous library, DBLPRO, is provided only to support Version 1.0 applications at run time only and is not a part of the tool kit.

2.1 APPLICATION DEVELOPMENT OVERVIEW

The major steps required for application program development are listed in Figure 2-1, followed by a list of the documentation appropriate to each of these steps.

It is important that you familiarize yourself with the Professional system and the P/OS operating system before proceeding beyond the preliminary installation activities. This is best done by following the Professional Tool Kit and Tool Kit DIBOL installation verification procedures and by reviewing the information contained in the installation documents and other related documentation.

PRELIMINARY ACTIVITIES

1. Preliminary installation activities
   - Professional Tool Kit software on host (if required)
   - Tool Kit DIBOL software on host pr P/OS
   - Professional hardware
   - Profesional resident libraries

2. Familiarize yourself with host, Professional system, and Tool Kit DIBOL.

3. Establish connection to host (if required) -- use menu selections

APPLICATION DEVELOPMENT

Application development on either a host system or P/OS.

4. Consider TOOL Kit DIBOL features and requirements.

5. Create and edit application.
   - Create menus (FDT).
   - Incorporate other system features; viz. FMS, GRAPHICS.

6. Compile Application; use DIB83P on host, DIBOL on P/OS.

7. Build Application; PAB on host, LINK on P/OS.

8. Write application command installation file.

9. Transfer task and installation command files to Professional -- use menu selections.

Continued on Page 2-3

APPLICATION INSTALLATION AND EXECUTION

Application installation and execution must take place on P/OS

10. Install application -- use menu selections.

11. Execute application.
    DDT is optional.
    If errors present, return to step 5 (edit).
    Build diskettes for distribution -- use menu selections.

Figure 2-1 Application Program Development Steps

The following is a list of the documents most useful for each of the program development steps (see the Preface for a complete list of documents related to the Professional):

Program Development Step(s)	Primary Documentation Source(s)
1,2	Professional and DIBOL Release Notes and Installation Guides
	Professional Owner's Manual
	Professional for Beginners: Hard Disk System
	Professional User's Guide for Hard Disk Systems
	Professional Tool Kit User's Guide
	This manual
3	Professional Communications Manual
4	This manual
	DIBOL-83 Compatibility Guide
5	Professional Tool Kit User's Guide
	Tool Kit DIBOL Language Reference Manual
	This manual
6	This manual (Chapter 3)

7	This manual (Chapter 4) Professional Tool Kit User's Guide
8	Professional Tool Kit User's Guide
9	Professional Communicatons Manual
10	Professional Tool Kit User's Guide
11	This manual (Chapter 6)
12	Professional Tool Kit User's Guide

2.2 PROFESSIONAL SYSTEM CONSIDERATIONS FOR DIBOL PROGRAMMERS

The best sources of Professional system information are the various Professional documents. Particularly useful are the **Professional Tool Kit User's Guide** and the **P/OS System Reference Manual.** References to the documentation is made throughout this chapter. Some of the characteristics of the Professional that are particularly important for the DIBOL programmer are presented in the following sections.

2.2.1 Interface to the User

One of the outstanding characteristics of the Professional system is its interface with the end user. Interaction is achieved through menus, help frames, and message frames. Facilities are provided so that a DIBOL program can take advantage of these standard interfaces. In addition, there is a facility to display system information.

2.2.2 Display

2.2.2.1 Terminal Addressing - The terminal is identified from within a DIBOL program with a "TT:" device designation. For the Professional, programs may also use "TI:". The printer is designated as "LP:".

2.2.2.2 Display Characters - The Professional display uses 8-bit characters instead of the 7-bit ASCII characters used with other DIGITAL terminals. The eighth bit accesses additional characters for other natural (human) languages. Therefore, if you want to generate and display the 8-bit characters necessary for these languages, you must use a Professional when creating programs; the professional is used as a remote terminal to a host system or as "host" system for program development.

The Professional accepts most VT100 escape sequences plus sequences that are peculiar to the Professional.

See Chapter 10 in the **Professional Tool Kit User's Guide** for the available characters and the **Terminal Subsystem Manual** for information on the characters and the escape sequences.

2.2.3 Keyboard

Several of the Professional keyboard function keys can be interpreted by the application at run-time. The GETKEY system service (see Chapter 8) provides the programmer with a way to avoid interpreting escape sequences.

2.2.4 P/OS Services and Facilities

The Professional system offers P/OS services and facilities that are accessed from within a DIBOL program via special DIBOL XCALLs that are documented in Chapters 7, 8, and 9.

The more important services and facilities are summarized below.

PRO/FMS-11

PRO/FMS-11 is used to create forms for use by an application for receiving user inquiries, receiving user responses, or displaying application output. The forms can be designed and modified on a screen using a forms editor and are stored in form libraries. Use of these forms reduces terminal I/O programming requirements. See Chapter 4 in the **Professional Tool Kit User's Guide** and Chapter 7 in this manual.

PRO/GRAPHICS

The P/OS operating system includes a library of graphics routines that the application developer can use to create graphic displays. The graphics primitives are callable from the application and include such features as graphics text and character attributes. See Chapter 4 in the **Professional Tool Kit User's Guide** and Chapter 9 in this manual.

FDT

The tool kit frame development tool allows the developer to create menus, messages, and help frames for the application user interface. See Chapters 4 and 12 in the **Professional Tool Kit User's Guide**.

RMS (file services)

Access to sequential, relative, and indexed files is provided by RMS-11 record management service routines and utilities. The RMS utilities available on the host system enable an application programmer to define, populate, update, and maintain files. There are no callable file services; RMS-11 is the file and record I/O interface between P/OS and the application. See Chapter 4 in the **Professional Tool Kit User's Guide.**

See the **DIBOL-83 Compatibility Guide** if you have DIBOL programs written with DMS file organization that you are migrating for execution on the Professional.

PRO/SORT

PRO/SORT is a general-purpose sorting program callable from the application. Its operation is similar to SORT-11. When PRO/SORT is invoked, a file containing the sort commands is passed from the application. See Chapters 4 and 11 in the **Professional Tool Kit User's Guide.**

NOTE

PRO/SORT does not support DIBOL decimal data types. Negative numeric fields cannot be properly sorted.

PROSE (editor)

PROSE is an end-user accessable text editor that uses the editing keys on the Professional keyboard. It can be used to create documents or to perform other related editing jobs. PROSE can perform these editing tasks as a part of an application as a callable editor task. See Chapter 11 of the **Professional Tool Kit User's Guide.**

Communications Facility

The P/OS communicatons facility operates in two roles: terminal emulation and file transfer between systems. It is available to the end-user through a menu. See Chapter 11 of the **Professional Tool Kit User's Guide.**

PRINT

File printing can be requested from the system menu. (See Chapter 11 in the **Professional Tool Kit User's Guide.**) In addition, printing is available from DIBOL programs via a subroutine. See Appendix C in this manual for a description of the PRINT external subroutine. The DIBOL LPQUE statement can also be used; however, only the filespec is recognized; the other arguments are ignored.

Messages

Messages can be accessed by the application from a specified message file by a call to the P/OS message service routines. See Chapter 13 of the Professional Tool Kit User's Guide.

2.2.5 Libraries

Object libraries are provided as a part of the DIBOL tool kit. These modules are the DIBOL external subroutines consisting of the UESL library (DBPLIB.OLB) and the OSSL library (DBPOSL.OLB). Subroutines from these libraries can be included in the application program as part of the development process.

The Professional utilizes libraries that are clustered; and in the case of RMS, they are also segmented. All the libraries are also resident; they share the same virtual address space on the Professional at run time. They are:

DIBOL run-time language support (DBLRES.TSK):

- the DIBOL language interpreter

NOTE

DBLPRO.TSK supplied for run-time support of V1.0 applications only.

Professional operating system services (POSRES.TSK):

- P/OS service routines (menu, message, help)

RMS support (RMSRES.TSK):

- file I/O support (record management services)

Optional Graphics support (CGLFPU.TSK or CGLEIS.TSK):

- Pro/Graphics support

2.2.6 Files

The Professional supports:

- Sequential, relative, and indexed files
- RMS file structure

The Professional does not support:

- ISAM utilities
- DMS files
- Multivolume files

An interactive ISAM utility is not supported by DIBOL on the Professional. You can use the RMS DEFine utility on host to create indexed application files for subsequent use by DIBOL programs or you can use the ISMCRE subroutine documented in Appendix C.

The DIBOL OPEN statement creates sequential files by default; OPEN O:R creates relative files. Indexed files cannot be created using the OPEN statement.

The use of OPEN O:P creates a file with special characters normally intended to be printed. See the **DIBOL-83 Language Reference Manual** for more detailed informaton about the OPEN statement.

2.2.7 DIBOL Debugging Technique (DDT)

The DIBOL Debugging Technique (DDT) operates on the Professional in the same manner as on other systems supporting DIBOL. In addition, the DDT display can be directed to a VT100 terminal connected to the printer port using a special cable. If the /B and /D options are used with the compiler to generate .ODL and .CMD files, DDT will be included automatically. See Chapter 6 of this manual for detailed information on DDT.

2.2.8 Error Messages (Condition Handling)

Applications may use the P/OS message facility to access and display messages from a file. All errors and messages generated by either the Professional system or by languages running on it make use of these files which are located separately from the program modules. A multiline traceback report is displayed by DIBOL when a nontrapped error occurs. The message remains displayed until the user presses the RESUME key.

See the **Professional Tool Kit DIBOL Message Manual** for Run-Time, DDT, and DIBOL compiler error messages.

2.3 DEVELOPMENT TASKS/STEPS

Figure 2-2 outlines the DIBOL program development tasks and the software used to accomplish these tasks.

APPLICATION DEVELOPMENT

APPLICATION DEVELOPMENT TAKES PLACE ON A HOST SYSTEM OR P/OS.

The DIBOL application (.DBL file) is produced by the user. The .DBL file includes the main program, data files, and external subroutines (if any).

1. Create application source files (.DBL).

2. Compile (DIB83P or DIBOL) with /B (and D). See notes 1 and 2 below.

3. Object files (.OBJ) result from step 2. .CMD files and .ODL files also included.

4. Build application with PAB (host) or LINK (P/OS). DDT module included if /D used. See notes 3 and 4 below.

5. Executable task (.TSK) results from step 4. DDT module may be present. See note 5 below.

APPLICATION INSTALLATION AND EXECUTION

APPLICATION INSTALLATION AND EXECUTION MUST TAKE PLACE ON P/OS

6. Create installation command file (.INS). Install with disk service from system menu.

7. Execute .INS and .TSK; Debug if DDT present.

NOTES

(1) /B causes automatic generation of .CMD and .ODL files; /D selects DDT.

(2) The .ODL file includes appropriate reference to .OBJ libraries and resident libraties.

(3) The object libraries are as follows: DBPLIB.OLB contains the UESL subroutines; DBPOSL.OLB contains OSSL subroutines and includes access to P/OS services from DIBOL

(4) Program segments to tie task to resident libraries described in note 5.

(5) The resident libraries available from the Professional Tool Kit on diskettes are as follows: DBLRES.TSK for DIBOL run-time system; POSRES.TSK for P/OS; RMSRES.TSK for file I/O; CGLEIS.TSL or CGLFPU.TSK for graphics; POSSUM for P/OS errror message files.

Figure 2-2 DIBOL Application Program Development Tasks

2.3.1 Coding/Editing

The creation of source code can be done using any editor that is available on the host system, or the editors currently available on the Professional (PROSE and EDT). Using the Professional has the advantage of allowing 8-bit characters to be included in the source programs. See the **DIBOL-83 Language Reference Manual** for detailed information.

2.3.2 Compiling

DIBOL programs are compiled with the DIBOL compiler as described in detail in Chapter 3 of this manual. Consider the various option switches to the compiler command line (note especially the /B build option and DDT considerations).

2.3.3 Application Building

Once the source program has been compiled it must be combined with other componets of the DIBOL program; user-developed external subroutines, UESL subroutines, OSSL subroutines. The physical manner in which the elements of the program will be stored and accessed is also determined by the way in which they are combined.

Certain operating characteristics such as logical unit and stack size specification are specified during application building. Application building is done on the host using the Professional Application Builder (PAB). See Chapter 4 in this manual and Chapter 5 in the **Professional Tool Kit User's Guide.**

2.3.4 Application Installation File

The application installation file identifies all the files and the hardware configuration for a given application. See Chapters 3 and 6 in the **Professional Tool Kit User's Guide.**

2.3.5 Transfer to the Professional System

If the application program is developed in the host environment, it is moved to the Professional before final debugging and execution. The transfer process is described in Chapter 7 of the **Professional Tool Kit User's Guide** and in Chapter 6 of the **Professional Communications Manual.** The program is transferred over a normal terminal line that has been connected to the Professional.

2.3.6 Installing/Executing/Debugging

The P/OS install routine (using the Application Installation File) identifies all executable images and data files, places the application name in the selected menu, and allows the application to be invoked by a menu selection. See Chapter 8 in the **Professional Tool Kit User's Guide** for application installation.

Execution is under control of DDT if it was compiled and linked with the application. All problems must be corrected either on the Professional or on the host system, if one was used. DDT output can be diverted to a VT100 connected to the Professional's printer port if DDT was specified in the compiler and application build commands. See Chapter 5 in this manual for detailed information on using DDT.

2.3.7 Creating Usable Diskettes

The final step in application development is to produce a diskette (or diskettes) that contains all the modules needed for a customer-usable package. This is accomplished with the Application Diskette Builder described in Chapter 9 of the **Professional Tool Kit User's Guide.**

2.4 PROGRAM MIGRATION

Program migration refers to moving DIBOL programs that are already developed for use on other systems to the Professional system. DIBOL as implemented on the Professional closely resembles DIBOL as implemented on CTS-300, CTS-500, and VAX. However, the Professional environment requires some differences because of functions and capabilities not available on other systems. The differences between the Professional system and the other systems supporting DIBOL and the new features available on the Professional system may require modification of existing programs before they are executable on the Professional.

The development cycle for migrated programs is the same as for programs developed for the Professional system; the editing phase is used to make necessary changes to the code for Professional system operation.

The differences in DIBOL implementation across the systems supporting the language is contained in the **DIBOL-83 Compatibility Guide.** That guide also contains an explanation of the how to transfer existing files to the host system. Take note of the subroutines and access to services that are documented in Chapters 7, 8, and 9 and in Appendix C of this manual.

CHAPTER 3

DIBOL COMPILER

The DIBOL compiler runs on the host operating system or on the Professional 300 system. It accepts DIBOL source code files that have been created by the user with an editor, and creates object files (.OBJ files) that are ready to be linked (application built) with other object modules to produce an executable DIBOL program. It can also produce command and overlay descriptor files to control the Professional Application Builder (PAB). The DIBOL compiler is invoked using the appropriate host command format.

A complete description of the DIBOL language is given in the DIBOL Language Reference Manual and the Introduction To DIBOL.

3.1 CHARACTERISTICS

3.1.1 Compiler Features

The DIBOL compiler can do the following:

- Read DIBOL source files and produce linkable object files.
- Accept up to six DIBOL source code files.
- Accept several qualifiers which govern the nature of the files produced, the listings, and the warning messages.
- Accept a qualifier that creates application builder files (.CMD and .ODL files).
- Respond to certain compiler directives in the source code. See the **DIBOL-83 Language Reference Manual** for details.
- Detect and report syntax errors and semantic errors.

The DIBOL compiler produces up to seven types of output.

- A listing of the source program (.LST file).
- A linkable file of compiled DIBOL data and statements (.OBJ file).
- A table of variable names used in the program (symbol table).
- A table of label names used in the program (label table).

- A report of the number and type of errors that were encountered during compilation.
- A command file.
- An overlay descriptor file.

The object file, the listing file, the contents of the listing file, the command file, and the overlay descriptor file are optional.

3.2 USING THE DIBOL COMPILER

3.2.1 Running the DIBOL Compiler

The manner in which the compiler is executed depends upon the environment in which program development takes place.

NOTE

The parameters (object, list, etc.) used in the compilation commands are described below after the various compiler commands are described.

If a host environment is used, the manner of compiler execution depends on the host operating system.

The compiler can be invoked on VAX in two ways:

- $MCR DIB83P<CR>
  *[object][,list]=source[,source,...]
- $MCR DIB83P [object][,list]=source[,source,...]

The compiler can be invoked on RSX (either RSX-11M or RSX-11M-PLUS) in three ways:

The form for execution is either of the following if the compiler is installed as ...DBP.

- >DBP [object][,list]=source[,source,...]
- >DBP <CR>
  *[object][,list]=source[,source,...]

The form for execution at any time:

- >RUN $DIB83P<CR>
  *[object][,list]=source[,source,...]

If the Professional 300 system environment is used, the manner of compiler execution is as follows.

1. Select the PRO/Tool Kit DIGITAL Command Language (DCL) from the menu.

2. The form for execution of the compiler is either of the following.

   - >DIBOL [/switch[/switch...]]source[,source...]

   - >DIBOL [/switch[/switch...]]<CR>
     File(s)?source[,source...]

The parameters used in the compiler commands are described in the following paragraphs.

object is the file specification in the general form:

dev:[directory]filnam.ext;ver[/switch[/switch...]]

- If dev: is not specified, the default device is the development system (host or PRO) default..
- If [directory] is not specified, the default directory is the development system default.
- If a file extension is not specified, then the default extension is .OBJ.
- If an object file is not specified, then an object file is not created.
- If a version number is not specified (;ver), the most recent version number is used.

,list is the listing file in the general form:

dev:[directory]filnam.ext;ver[/switch[/switch...]]

- If dev: is not specified, the default device is the current system default. Often the listing file is directed to the terminal (TT: on VAX, TI: on RSX) by specifying the device. This could also be used with just a file name to store the listing on a disk.
- If [directory] is not specified, the default directory is the development system default.
- If a file extention is not specified, then the default extension is .LST
- If a listing file is not specified, than none is produced; if only an object file is specified, only that file is generated.

- If only a listing file is desired with no object file, then specify only a listing file specification preceded by the comma. The comma indicates the omission of the object file.

source — is the DIBOL source file supplied as input to the compiler. Source files are specified in the general form:

```
dev:[directory]filnam.ext;ver[switch[/switch...]]
```

- If dev: is not specified, the default device is the development system (host or PRO) default.
- If [directory] is not specified, the default directory is the development system default.
- If a file extention is not specified, the default extension is .DBL.
- If a version number is not specified (;ver), then the most recent (highest version number) version is used.
- If multiple input files are specified, a single object and/or listing file is produced as output as though the input files had been combined (concatenated) before the compilation had begun.

switch — is the compiler command qualifier. Qualifiers are discussed in detail in the following section. If more than one qualifier is specified, they are separated by slashes. When multiple qualifiers are specified, their order is unimportant.

3.2.2 Command Qualifiers

Command qualifiers (switches) restrict or modify the action specified by the command. The qualifiers are listed below for compilation on a host system (HST) and the Professional system (P/OS).

The qualifier is appended to the DIBOL command (DCL only) or a file specification (MCR or DCL) with a forward slash (/) as follows.

```
/switch
```

where:

/ — specifies that a command qualifier or file specification qualifier follows.

switch — specifies the qualifier to be activated.

The qualifiers used with the HOST are of the form /switch, where switch is a single alphabetic character that specifies a particular action. If a particular qualifier is not used, the specified action is not activated (defaulted).

The qualifiers used with P/OS are of the form /[NO]switch, where switch is a POSITIVE WORD that specifies a particular action and NOswitch is the NEGATIVE that specifies deletion or omission of that action. If a particular qualifier is not used, its P/OS default value is used; that default value is shown in the table under ACTION.

Only the number of letters that uniquely identifies the qualifier need be used. For example, /L is sufficient for /LISTING and /OB is sufficient for /OBJECT (/O could be /OBJECT or /OPTIMIZE).

QUALIFIER		ACTION
HST	P/OS	
/B	/[NO]BUILD	directs the compiler to automatically invoke a build procedure that creates command (.CMD) files and overlay descriptor (.ODL) files for direct input into the application builder. The P/OS default value is /NOBUILD. If you want an overlaid DIBOL program, you will have to modify the overlay descriptor file created by the compiler. See the **RSX-11M/M-PLUS Task Builder Manual** and Chapter 4 of this manual.
/D	/[NO]DEBUG	directs the compiler to include debugging information in the object module. This debugging information is required if the DIBOL Debugging Technique is to be used to debug this module. If the /B and /D (or /BUILD and /DEBUG) qualifiers are used together, the overlay descriptor file (.ODL) will include the required reference to DBLDDT.OBJ. The P/OS default value is /NODEBUG.
/O	/[NO]OPTIMIZE	directs the compiler suppress tracking of line numbers as part of the object code and causes other optimizations to occur. The resulting program code is smaller and executes more quickly. Line numbers are required by DDT for single-step execution and breakpoints. Therefore, /D (or /DEBUG) overrides /O (or /OPTIMIZE). If /O (or /optimize) is not used, line numbers tracking occurs in the object module and optimization does not occur. The P/OS default value is /NOOPTIMIZE

/P=n	/PAGE=N	directs the compiler to use the argument "n" as the page length for the listing file. The specified value includes three lines for the top margin, three lines for a header, and three lines for the bottom margin (nine in all). If this qualifier is omitted, the system default lines per page is used.
/N	/[NO]STANDARD	directs the compiler to include non-standard features. This qualifier must be used if block mode I/O is utilized. The P/OS default value is /STANDARD.
/S	/[NO]TABLE	directs the compiler to generate a symbol table and a label table as part of the listing file. The P/OS default value is /NOTABLE.
/W	/[NO]WARNING	directs the compiler to suppress warnings during compilation. The P/OS default value is /WARNING.
--	/LIST[=filnm]	creates a listing file with default file type LIS. If no file (filnm) is specified, the first file specification in the command line is used. If the negative form is used, no listing is produced. The P/OS default value is /NOLIST. There is no HST form of this command.
--	/[NO]OBJECT[=filnm]	creates an object file with default file type OBJ. If no file (filnm) is specified, the first file specification in the command line is used. If the negative form is used, no object file is produced. The P/OS default value is /OBJECT. There is no HST form of this command.

3.2.3 Specifying Output Files

During the early stages of program development, you may find it helpful to specify no object file which will suppress the production of object files until your source program compiles without errors. The /S (or /TABLE) qualifier produces information that helps in debugging.

3.2.4 Contents of the Listing File

The listing file consists of the following components: the program listing and, if selected, the symbol table and label table. The contents of each component of the listing file are described in the following paragraphs.

3.2.4.1 THE PROGRAM LISTING - The program listing consists of the original source code with line numbers prefixed to each DIBOL statement. Line numbers are not assigned to the following:

The compiler directives: .ENDC, .IFDEF, .IFNDEF, .INCLUDE, .LIST, .NOLIST, .PAGE, .TITLE

Continuation lines

Blank lines

Comment lines

Line numbers are used in the label table to identify the location of variable names and label names. In addition, line numbers are useful in debugging (DDT) and error trapping at run time.

For each error detected during compilation, an error message appears in the line-numbered listing. Each message appears immediately after the line in which the error is detected.

Example of an error reported in the listing:

```
     3    PROC
     4    L1,        GOTO(L1,L2,L3),VAR
%DIBOL-W-LABOUTBLK, Label out of context block; L2
     .
     .
     .
```

The above example shows a warning message. An error message would have a "-E-" in place of the "-W-."

3.2.4.2 The Symbol Table - The symbol table contains descriptive information about each variable in the source code. The table consists of four columns with the following headings: NAME, TYPE, DIMENSION, SIZE. The information under each heading is described in Table 3-1.

3.2.4.3 The Label Table - The label table contains descriptive information on each label name and external subroutine name in the source code. The table consists of three columns with the following headings: NAME, TYPE, LINE #. The information under each heading is described in Table 3-2.

TABLE 3-1

CONTENTS OF THE SYMBOL TABLE

COLUMN HEADING	DESCRIPTION
NAME	A list of data field names (RECORD, COMMON, field) used by the program.
TYPE	The data type of the field. The data type may be alpha (ALPHA), numeric (DECIMAL), or improper definition (IMDEF). If the field is defined in a COMMON statement, its data type identification is preceded with the symbol "C-"; C-DECIMAL, for example.
DIMENSION	The number of data elements in the field. A dimension of zero is assigned to the data type IMDEF.
SIZE	The number of characters required to store the field. If the field is an array, the size of one array element is shown.

TABLE 3-2

CONTENTS OF THE LABEL TABLE

COLUMN HEADING	DESCRIPTION
NAME	The name of each label and external subroutine that is used by the program.
TYPE	The type is either a statement label (LABEL), or an external subroutine (EXSUB).
LINE #	The line number at which the label is defined. Zero is assigned to all improperly defined or referenced label names and external subroutine names.

3.2.5 Error Messages

The errors detected by the compiler are errors of syntax or semantics in specific DIBOL statements. These compiler errors should be distinguished from run-time errors, which are detected during program execution.

The compiler error messages are described in the **Professional Tool Kit DIBOL Message Manual.**

3.2.6 Example DIBOL Program and Program Listing

The following is a source listing of the program PAYRL1.DBL. It computes regular and overtime gross pay (note that the listing has been modified to fit an 80-column page:.

```
RECORD PERSON
    HOURPY,   A3,'100'              ; Hourly payrate of $1.00.
    HOURWK,   12A2,'39','40','41',
&                  '42','43','44',
&                  '45','46','47',
&                  '48','49','50' ; An array of hours worked per week
RECORD TEMP                         ;
    A,    D2                        ; Working variable for HOURWK
    B,    D3                        ; Working variable for HOURPY
    K,    D2                        ; Loop counter -- index
    C,    A7                        ; Display weekly pay
    PAY,  D10                       ; Field to contain gross pay amount
PROC                                ;
    OPEN(1,O,'TT:')                 ; Open the terminal for output
    DISPLAY(1,'HOURS','    ',
&             'GROSS PAY',13,10)    ; Display the output headings
    B = HOURPY                      ; Work variable equals hourly pay
    FOR K FROM 1 THRU 12            ; Compute for each field in array
        BEGIN                       ;
        A = HOURWK(K)               ; Work variable equals array field
        PAY = A * B                 ; Compute regular pay
        IF (A .GT. 40)
&       PAY = PAY + (B*(A-40)/2)    ; Compute if overtime pay
        C = PAY, '$$$$.XX'          ; Format the pay as dollars
        DISPLAY(1,' ',HOURWK(K),
&               '    ',C,13,10)     ; Display the answer on the terminal
        END                         ; End loop
    CLOSE 1                         ; Close the terminal
    END                             ;
```

The following file specification is used to compile this source file and produce an object file and listing file with a symbol table (there are no labels in this example source file):

```
*PAYRL1.OBJ,PAYRL1.LST=PAYRL1.DBL/S
```

As a result of this command, the source program is compiled and the following items are written to the file PAYRL1.LST: a line-numbered listing and a symbol table. The contents of PAYRL1.LST is shown below.

```
PAYRL1              15-APR-1983 14:36:44      PRO DIBOL-83 V1-05   Page   1

     Data  Division              DRC4:[WRITER.DICKR.PRODIB17]PAYRL1.DBL;1


     1  RECORD PERSON
     2      HOURPY,   A3,'100'                ; Hourly payrate of $1.00.
     3      HOURWK,   12A2,'39','40','41',
        &                 '42','43','44',
        &                 '45','46','47',
        &                 '48','49','50'     ; Hours worked per week
     4  RECORD TEMP                           ;
     5      A,   D2                           ; Working variable for HOURWK
     6      B,   D3                           ; Working variable for HOURPY
     7      K,   D2                           ; Loop counter -- index
     8      C,   A7                           ; Display weekly pay
     9      PAY, D10                          ; Field for gross pay amount


PAYRL1              15-APR-1983 14:36:44      PRO DIBOL-83 V1-05   Page   2

     Procedure  Division         DRC4:[WRITER.DICKR.PRODIB17]PAYRL1.DBL;1


    10  PROC                                  ;
    11      OPEN(1,O,'TT:')                   ; Open the terminal for output
    12      DISPLAY(1,'HOURS','    ',
        &            'GROSS PAY',13,10)       ; Display the output headings
    13      B = HOURPY                        ; Work variable equals hourly pay
    14      FOR K FROM 1 THRU 12              ; Compute for each field in array
    15           BEGIN                        ;
    16           A = HOURWK(K)                ; Work variable equals array field
    17           PAY = A * B                  ; Compute regular pay
    18           IF (A .GT. 40)
        &        PAY = PAY + (B*(A-40)/2)     ; Compute if overtime pay
    19           C = PAY, '$$$$.XX'           ; Format the pay as dollars
    20           DISPLAY(1,' ',HOURWK(K),
        &                '     ',C,13,10)    ' Display answer on terminal
    21           END                          ; End loop
    22      CLOSE 1                           ; Close the terminal
    23      END                               ;
No errors detected
```

Symbol Table DRC4:[WRITER.DICKR.PRODIB17]PAYRL1.DBL;1

Name	Dim	Type	Size
PERSON		Alpha	27
HOURPY		Alpha	3
HOURWK	12	Alpha	2
TEMP		Alpha	24
A		Decimal	2
B		Decimal	3
K		Decimal	2
C		Alpha	7
PAY		Decimal	10

CHAPTER 4

THE APPLICATION BUILDER

The Application Builder performs the final step in preparing a DIBOL program for execution. It combines object modules created by the DIBOL compiler with object modules extracted from the object libraries on into an executable program (task) and resolves references to resident libraries. Every module produced by the DIBOL compiler must be processed by the application builder before it can be executed.

If program development takes place on a host system using Professional Host Tool Kit DIBOL, the Professional Application Builder (PAB) is used. PAB is an enhanced version of the RSX-11M and RSX-11M-PLUS task builders.

If program development takes place on the Professional using PRO/Tool Kit DIBOL, the application builder is called LINK.

NOTE

> The term LINK is used throughout this chapter. If development was done on the host, substitute PAB for LINK.

The application builder does the following:

- Combines (links) external subroutines and the main program(s) into an executable task.
- Combines the main program(s) and external subroutines with modules from the appropriate object libraries.
- Resolves references to resident libraries.
- Creates overlay structures.
- Produces a load map that shows the layout of a program in memory.
- Produces error messages in the event of an inability to build the task as specified (see the **RSX-11M/M-PLUS Task Builder Manual**).

This chapter presents the more common decisions a DIBOL programmer faces during application building. It is an overview of a process that can be quite complex. See the **RSX-11M/M-PLUS Task Builder Manual** for a complete discussion of application building.

4.1 LIBRARIES

The libraries supplied for the application builder are provided solely for linkage; modules from these libraries (UESL and OSSL) are incorporated into the task image by LINK. Application building does not produce a task image that is executable on the host; a run-time system is required to interpret and execute the code linked by the application builder.

NOTE

> The application builder for Professional programs (PAB or LINK) translates all references of LB:[1,1] to LB:[1,5] for input files to avoid conflict with other libraries on the host. All required libraries for the Professional are in LB:[1,5] on the host.

The Professional system itself utilizes resident, clustered, (and segmented in the case of RMS) libraries that are prebuilt to be used as part of the Professional system.

The libraries are:

Supplied with the DIBOL Tool Kit:

1. The Universal External DIBOL Subroutine Library (DBPLIB.OLB) -- This provides common DIBOL external subroutines (DATE, TIME, etc.)

2. The Operating System Specific DIBOL Subroutine Library (DBPOSL.OLB) -- This provides system specific DIBOL external subroutines.

Supplied with the Professional Tool Kit:

1. The RMS Resident Library symbol table and truncated task image files (RMSRES) -- These supply information necessary to link the DIBOL task with the RMS resident library.

2. Graphics -- There are two supplied graphics libraries: CGLFPU and CGLEIS.

3. P/OS Service Routines (POSRES) (eg., menu, help, text) -- These supply information necessary to link the DIBOL task with the P/OS services.

4. The POSSUM system services -- These are actually integrated into the P/OS executive but if any of the POSSUM facilities are to be used by a DIBOL program, POSSUM must appear in the "CLSTR=" line of the .CMD file as though it were a discrete resident library.

5. Callable system utilities (eg., SORT, FMS, and EDIT).

6. The DIBOL Resident Libraries for V1.6 (DBLRES) -- These supply information necessary to link the DIBOL task with the RMS resident library and support the application at run time.

7. The DIBOL Resident Libraries for V1.0 (DBLPRO) -- These support V1.0 applications at run time.

4.2 APPLICATION-BUILD PROCEDURE DECISIONS

NOTE

Application building a DIBOL program requires the use of both a command file and an .ODL file. In particular, you should not application build a program without using an .ODL file with the references to RMS which are in the compiler generated (/B) .ODL file. While such an approach may succeed, it is not supported and is unlikely to be compatible with future releases of RMS. The options supplied by the use of the /B switch in the .CMD file will also be needed.

4.2.1 Generating the Command and Descriptor Files

When compiling a main program, use the DIBOL compiler build (/B) option to generate a command file (.CMD) and overlay descriptor file (.ODL) for your application. LINK uses these files to define how libraries are referenced, and to specify special-purpose buffers, logical unit numbers (LUNs), and event flags (EFNs).

Look at the .CMD and .ODL files in the text and examples. You may need to edit the command file or overlay descriptor file for your particular application. The following sections describe the information that must be contained in these command and overlay descriptor files.

4.2.2 Editing the Command (.CMD) File

A command file output by the Tool Kit DIBOL compiler for an application named "TEST1" would look like this:

```
SY:TEST1/CP=SY:TEST1/MP
UNITS=24
ASG=TI:16                   ; Foreground terminal
ASG=TI:17                   ; DDT terminal
ASG=SY0:18                  ; Channel for RENAM/DELET
ASG=TI:19                   ; POSRES Terminal
ASG=SY:23                   ; Directory Searches
ASG=LB:24                   ; Message board
;
CLSTR=DBLRES,POSRES,POSSUM,RMSRES:RO
;
TASK=TEST1
```

The command file would be followed by extend section and global definition lines (shown in later examples).

The command file automatically references the DIBOL RTS and the required P/OS service routine library clustered with RMS. You should edit the command file if you want to reference any libraries (such as Graphics) other than the language RTS (DBLRES), the system services libraries (POSRES, POSSUM), and RMS (RMSRES).

Note the following points concerning the .CMD file:

1. Clusterable Libraries

   List all required clusterable libraries on the line beginning "CLSTR". Note that the default library must be the Tool Kit DIBOL language RTS. For example:

   ```
   CLSTR = DBLRES,POSRES,POSSUM,RMSRES:RO
   ```

   This line identifies the DIBOL (RTS) (DBLRES) as the default library and RMS and POSRES as referenced libraries. All libraries are read-only; therefore, the :RO switch is always required.

2. Additional Logical Units.

   DIBOL requires the first 18 logical units. CTAB services may require an additional five for a total of 23. The total number must be declared in the UNITS statement, otherwise LINK will default to six. The DIBOL BUILD option automatically specifies 24.

3. Extend Section Commands

If your application uses P/OS user interface services in the POSRES library, allocate buffer space with extend section (EXTSCT) commands in the command file. The help and menu user interface routines access definition files created with FDT. Definition files contain frames with fields of information. For each type of service, allocate a buffer that is large enough to accomodate the largest frame of that type used by the application. Frame size is calculated by adding together the sizes of all the fields in a frame, then adding an overhead of eight bytes for each field in a frame plus 20 bytes per frame. See the **Professional Tool Kit User's Guide**, Chapters 4 and 5, for complete information on definition files, frames, fields, and user interface services.

NOTE

Version 1.0 users should note that for V1.5 OR LATER the EXTSCT for MS$BUF message facility is no longer used.

4. Logical Unit Numbers

P/OS user interface services require logical unit numbers (LUNs) to perform file I/O to menu, help, and message definition files. LUN assignments for P/OS I/O can be made with global definition commands (GBLDEF) in the command file. In assigning LUNs, remember to account for LUNs needed by the DIBOL Language RTS (18). You must include the LUN for DIBOL RTS messages:

```
gbldef = ms$lun:25      ; message frame file
```

If one type of P/OS service is never used, such as help services, specify 0 in the global definition line for help files. Do not omit any global definition line. After assigning LUNs, determine the total number needed and include that number in the UNITS command.

NOTE

All required LUNs are included by DIBOL. If you want DDT direct output to the other (debugging) terminal, change ASG = TI:17 to ASG = TT2:17.

5. Event Flags

P/OS user interface routines also require an event flag (EFN) to perform terminal I/O. The event flag assignment should not conflict with event flag assignments made for other purposes by the application. Language RTS routines use DIGITAL reserved event flags (flags 25-32). The event flag assigned with the TT$EFN line should not be a DIGITAL reserved flag. The DIBOL BUILD option assigns EFN 1.

IMPORTANT NOTE

The assign (ASG) commands and units list DECIMAL numbers. Extend (EXTSCT) and global definition (GBLDEF) lines specify OCTAL numbers.

The following list shows the extend section and the global definition lines that are added below the "CLSTR" line in the command file. The values for the extend section lines (EXTSCT) are calculated for the largest possible frame of each type. That is, if every field on a menu, help, and message frame were filled with the maximum amount of data, the respective buffers would have to be allocated the sizes shown.

```
                                ; DEFINE BUFFER SIZES
extsct = mn$buf:4540            ; static single-choice menu
extsct = dm$buf:4540            ; dynamic single-choice menu
extsct = mm$buf:1000            ; multi-choice menu
extsct = hl$buf:3410            ; help text/menu
extsct = fl$buf:4310            ; file selection/specification
;
gbldef = tt$lun:23              ; (19.) terminal I/O
gbldef = hl$lun:24              ; (20.) help frame file
gbldef = ms$lun:25              ; (21.) message frame file
gbldef = mn$lun:26              ; (22.) menu frame file
gbldef = wc$lun:27              ; (23.) directory searches
gbldef = mb$lun:30              ; (24.) message board lun
;
gbldef = tt$efn:1               ; terminal I/O event flag
```

NOTE

DIBOL includes these buffers with maximum allocation. Keep only the required buffers. Memory used for these buffers is available to the task. If a service is not to be used, set extsct to zero.

Example:

```
extsct = mn$buf:0
extsct = dm$buf:0
extsct = mm$buf:0
extsct = hl$buf:0
extsct = fl$buf:0
```

See also Chapters 4 and 5 of the **Professional Tool Kit User's Guide.**

4.2.3 Editing the Overlay Descriptor File

The Tool Kit DIBOL compiler would produce this overlay descriptor file for the tool kit application TEST1:

```
                .ROOT     APOBJ$-DBLIB$-RMS$
APOBJ$:         .FCTR     TEST1
DBLIB$:         .FCTR     LB:[1,5]DBPLIB/LB-LB:[1,5]DBPOSL/LB
RMS$            .FCTR     RMSROT
@LB:[1,5]RMSRLX.ODL
                .END
```

If you want to reference any Tool Kit object libraries facilities, such as FMS-11, you must edit the overlay descriptor file. Chapter 7 illustrates the required edits for FMS-11 support.

4.3 APPLICATION-BUILDING DIBOL PROGRAMS

Use LINK to create an application task image (.TSK). See the **Professional Tool Kit User's Guide** for complete information on Professional Application Builder Commands. The remainder of this chapter contains information specific to the DIBOL developer.

4.3.1 A Simple Application Build

A simple application build can be done using the .CMD and .ODL files created by use of the /B qualifier in the compiler command line. To use this method, respond to the "DIBOL BUILD>" prompt with a carriage return to initiate the generation of the .CMD and .ODL files. If DDT is desired in the application, include the /D qualifier in the compiler command line and DBLDDT.OBJ will be automatically inserted in the .ODL file.

4.3.2 Background Information

There are two items in particular that influence the application build procedure: the resident libraries and overlaying. A brief discussion of these two items follows. After this discussion, examples illustrate the various options.

NOTE

> Linking with POSRES removes approximately 4KW of user space. POSRES is required because it is used for run-time error messages. Use of the /B generated command file will result in maximum options being chosen for system services. You can remove unused services for more user space.

One way to deal with a memory space problem is to overlay user external subroutines. The DIBOL compiler, with the use of the /B switch, provides an .ODL file that can serve as a prototype to be modified by the user.

The RMS resident library, RMSRES, occupies approximately 23K words in main memory. However, no more than 8K words of user space are used by RMS code (RMSRES) at any given time. Mapping the necessary RMS code into user space occurs very rapidly since it takes place within main memory. This is in contrast to disk-resident overlays in which a module must be moved from disk memory to main memory. You must use the RMS resident library. If there is not enough room in memory for all segments, they will be swapped on and off disk with the normal overhead of disk swapping.

For more information on overlays, see the **RSX-11M/M-PLUS Task Builder Manual.**

4.3.3 Application Builder Error Messages

The application builder produces diagnostic and fatal error messages. Some diagnostic error messages merely tell you about an unusual condition. If you consider the condition normal to your task, you can run the task image.

In general, you can application build and execute programs that have compiler errors; this is not recommended. If the error occurs in an overlaid subroutine, the application builder will report an illegal format error and will not link in the subroutine. The resulting executable task is unpredictable. If such a program is run, a message is displayed prior to program execution indicating that the program has compilation errors.

If the task exceeds memory limits at execution time, you will need to reduce some of the values provided by the compiler-generated .CMD and .ODL files (maximum values are the default). You could also try compiling with the /O option or using additional overlays. A frequent cause is having a large number of files open at one time; especially if those files use bucket sizes greater than one. If memory space is a problem, a map listing should be generated using LINK by specifying a second output file to determine where the space is being used. Keep in mind that extra space is required at run time for the run-time system and for each open file. At execution time 48 KB is available for each task exclusive of the clustered libraries.

For a listing of the application builder error messages, see the **RSX-11M/M-PLUS Task Builder Manual.**

4.3.4 Command Format

A command to the application builder consists of two parts. In the first part you specify the output files (task and, optionally, a map), the input files (object modules and subroutine libraries, from an overlay description file which replaces/specifies input files only), and one or more switches that prescribe functions to be performed. In the second part you specify the appropriate options, for example, which resident libraries to use.

The command for an existing .CMD file (TEST):

```
LINK @TEST
```

where TEST is the .CMD file. The .CMD file generated by the DIBOL compiler looks like the following example for program PROG (this would be the contents of "@TEST").

```
PROG/CP=PROG.ODL/MP                                    File Specification
UNITS=24                                                        \
ASG=TI:16       ; Foreground terminal                           |
ASG=TI:17       ; DDT terminal                                  |
ASG=SY0:18      ; Channel for RENAM/DELET                       |
                                                                |
ASG=TI:19       ; POSRES terminal                               |
ASG=SY:23       ; Directory Searches                            |
ASG=LB:24       ; Message board                                 |
;                                                               |
CLSTR=DBLRES,POSRES,POSSUM,RMSRES:RO                            |
;                                                               |
TASK=TEST1                                                      |
;                                                               |
extsct  = mn$buf:4540    ;static single choice menu             |
extsct  = dm$buf:4540    ;dynamic single choice menu            |
extsct  = mm$buf:1000    ;multi-choice menu                     \
extsct  = hl$buf:3410    ;help text/menu                          Options
extsct  = fl$buf:4310    ;file selection/specification          /
;                                                               |
gbldef  = tt$lun:23      ; (19.) terminal I/O                   |
gbldef  = hl$lun:24      ; (20.) help frame file                |
gbldef  = ms$lun:25      ; (21.) message frame file             |
gbldef  = mn$lun:26      ; (22.) menu frame file                |
gbldef  = wc$lun:27      ; (23.) directory searches             |
gbldef  = mb$lun:30      ; (24.) message board lun              |
;                                                               |
gbldef  = tt$efn:1       ; terminal I/O event flag              |
;                                                              /
//
```

The following is an explanation of some of the components of the file specification section of the .CMD file:

PROG/CP — represents the file specification of the final executable program (the task file) that is to be created. The default extension is .TSK.

NOTE

The /CP switch is always required with the task file for Tool Kit DIBOL.

PROG.ODL/MP — represents the file specification of the overlay description file. The assumed extension is .ODL (for an overlay description file).

NOTE

The use of an .ODL file is mandatory for Tool Kit DIBOL. Therefore, the /MP switch (see below) is required with the .ODL file specification.

Switches — are appended to the file name. For more advanced switch functions, see the RSX-11M/M-PLUS Task Builder Manual.

Commonly Used Application Builder Switches

Mnemonic	Function
/CP	Always required on the task file (.TSK).
/LB	Specifies that the input file is a subroutine object library file (.OLB).
/MP	Specifies that the input files are described in a separate overlay description file (.ODL)
/WI	Instructs LINK to produce a wide MAP listing file (.MAP).

Resident libraries are clustered. The form is:

```
CLSTR=DBLRES,POSRES,POSSUM,RMSRES:RO
```

4.3.5 Files used by the Application Builder

Files specified as input files to the application builder fall into four categories:

1. COMMAND and OVERLAY DESCRIPTION LANGUAGE FILES (.CMD and .ODL) - These files contain instructions for LINK. Both of these files are generated by the DIBOL compiler when the /B switch is used.

   The command file refers LINK to the .ODL file and specifies the LINK options.

   The overlay description file that describes the structure of the overlays, or the way that the input (object) files are to be combined, must be created and supplied to the application builder. The .ODL file identifies the .OBJ and .OLB files (numbers 2 and 3, following).

   NOTE

   DIBOL always uses an .ODL file specified in the .CMD file and this .ODL file refers to the above input files.

2. OBJECT MODULE FILES (.OBJ) - These are files created by the DIBOL compiler (DIB83P). They have a default extension of .OBJ and are the files that are to be linked together in order to make the executable program (that is, the task file).

3. LIBRARY FILES (DBPLIB.OLB, DBPOSL.OLB, and USER.OLB) - These files are searched for .OBJ modules to be included with the .OBJ file(s) in 2, above.

   The DIBOL subroutine library files are DBPLIB.OLB and DBPOSL.OLB.

   USER.OLB refers to libraries that you create with the Librarian (see the operating system documentation). They will probably contain frequently used subroutines that you have placed in the library for easy access by one or more of your programs.

4. REFERENCES TO RESIDENT LIBRARY FILES (xxx.TSK and xxx.STB) - These are files used by LINK to resolve RESLIB (resident library) entries. The resident libraries follow.

   - The Tool Kit DIBOL resident library file (DBLRES).
   - The RMS resident library file (RMSRES).
   - The P/OS operating system (POSRES, POSSUM).
   - Other resident libraries possibly required for optional features.

Input files may be entered as complete file specifications or as file names alone. The extension .OBJ is assumed except where the /LB switch is used; then the assumed extension is .OLB, or where the /MP switch is used, an .ODL file.

Subroutine libraries are used by the application builder to resolve subroutine references in the object modules. If a subroutine name is not found in the input files or any of the libraries, the application builder displays an error message on the terminal.

An overlay description file must be the only input file specified. This file must describe all .OBJ and .OLB files and it must define the structure of the overlays (see Section 4.7).

The application builder outputs two types of files for DIBOL programs:

1. TASK FILE - The task file is a program in its final executable form. The application builder gives it a .TSK extension.

2. MAP FILE - This is an optional file that produces a listing of the task file's memory allocation. The application builder outputs this file with a .MAP extension.

4.4 EXAMPLES OF .ODL FILES

4.4.1 Without Overlays

The application-build procedure that links PRGRM2 and USERLIB against the RMS resident library follows. No map file is produced.

```
.ROOT     APOBJ$-DBLIB$-RMS$
APOBJ$:   .FCTR     PRGRM2
DBLIB$:   .FCTR     USERLIB/LB-LB:[1,5]DBPLIB/LB-LB:[1,5]DBPOSL/LB
RMS$:     .FCTR     RMSROT
@LB:[1,5]RMSRLX.ODL
          .END
```

4.4.2 With Overlays

The overlay description file PGM.ODL that is used in the application-build procedure for program PGM with various subroutines (SUB's) is:

```
.ROOT    PGM-LIB2-LIBR-RMS$-*(A,B,C,D,E,F,G)
A:   .FCTR     SUB01-LIBR
B:   .FCTR SUB11-H
C:   .FCTR SUB12-LIBR-*(J,SUB21-LIBR,SUB03-LIBR,SUB04-LIBR)
D:   .FCTR SUB13-*(SUB23-LIBR,SUB32-LIBR,SUB04-LIBR,I,J)
E:   .FCTR SUB14-*(J,SUB22-LIBR,SUB03-LIBR,SUB04-LIBR)
F:   .FCTR SUB16-*(J,SUB22-LIBR,SUB03-LIBR,SUB04-LIBR)
G:   .FCTR SUB18-*(J,SUB22-LIBR,SUB03-LIBR,SUB04-LIBR)
H:   .FCTR LIBR-*(J,SUB21-LIBR,SUB22-LIBR,SUB03-LIBR,SUB04-LIBR)
I:   .FCTR SUB06-LIBR
J:   .FCTR SUB31-LIBR
LIB2:   .FCTR FGLIB/LB:USB03:USB06:USB09:USB10:USB11:SUB02:SUB05:PGMLG-LIBR
LIBR:     .FCTR     LB:[1,5]DBPLIB/LB-LB:[1,5]DBPOSL/LB
RMS$:     .FCTR     RMSROT
@LB:[1,5]RMSRLX.ODL
          .END
```

CHAPTER 5

RMS-11 AND DIBOL ON THE PROFESSIONAL

Record Management Services (RMS-11) used on the Professional is a general purpose file processing environment which provides data storage, retrieval, and modification facilities. When executing DIBOL application programs on the Professional, file processing is ultimately done by RMS-11.

NOTE

For further information on all of the topics covered in this chapter, see the RMS-11 manuals listed in the Preface.

This chapter briefly explains the RMS-11 file organizations and access modes. More information on access is contained in the **DIBOL-83 Language Reference Manual.**

An indexed file is created on a host system with the RMS facilities or the DIBOL OSSL subroutine ISMCRE. Only the ISMCRE subroutine can be used with the Professional 300 System. This chapter provides an example of how to create an RMS indexed file on the host for use with dibol on the Professional.

5.1 RMS-11 FILE ORGANIZATION

RMS-11 provides three file organizations, each of which is discussed in this chapter:

1. Sequential file

2. Relative file

3. Indexed file

The organization of a file establishes the techniques you can use to retrieve and store data in that file. These techniques are known as access modes. The access modes available under Professional Tool Kit DIBOL are:

1. Sequential access

2. Random access

In addition, block mode I/O can be used to read and write entire blocks of data rather than transferring a record at a time.

The organization and characteristics of a file are specified when an RMS file is created. The file characteristics are called attributes. Among those you can specify are the storage medium, the file name and protection specifications, the record format and length, and the file allocation information.

5.1.1 Sequential File Organization

In the sequential file organization, records appear in the order in which they are inserted. That is, records can only be retrieved from a file in the same order in which they were originally written, and there is no attempt to optimize file access. Therefore, you will have to scan the file from the beginning to locate a record. Records may be added only to the end of a sequential file, and records within the file cannot be deleted.

Sequential files are created by a DIBOL program when a file is opened using the OPEN statement in O mode without any submode specifier.

5.1.2 Relative File Organization

The relative file organization permits a file's records to be accessed randomly, based on their positions relative to the beginning of the file. The records in a relative file may also be retrieved sequentially, by successively requesting the next record. However, the relative organization also permits access, by record number, to a record anywhere within the file without previously accessing any other record.

Relative files can be created by a DIBOL program when a file is opened using the OPEN statement with the O:R submode specifier (see the **DIBOL-83 Language Reference Manual**). Relative files can also be created with the RMSDEF utility.

5.1.3 Indexed File Organization

The indexed file organization permits the random access of records based on the values of fields contained within the records. The location of these records is transparent to the program. RMS-11 controls the placement of records in this type of file; the contents of key values within the records govern this placement.

An indexed file is created with the RMS facilities or the DIBOL OSSL subroutine ISMCRE. Only the ISMCRE subroutine can be used with the Professional 300 System. For an example of RMSDEF, see Section 5.6 in this manual. For more information on RMSDEF and RMS see the RMS manuals listed in the Preface. The ISMCRE external subroutine is described in Appendix C of this manual.

5.1.3.1 Indexed File Keys - For an indexed file, you must define at least one key, the primary key. With RMS-11, however, you may also define alternate keys, up to a total of 255 keys (1 primary and 254 alternate keys). As programs write records into an indexed file, RMS-11 locates the values contained in the primary and alternate key positions of each record. From these values, RMS-11 builds a tree-structured table known as an index. An index consists of a series of entries, each containing a key value copied from a record that a program wrote into the file. With each key value is a pointer to the location in the file of the record from which the value was copied. RMS-11 builds and maintains a separate index for each key defined for the file. When alternate keys are defined, RMS-11 builds and stores an additional index for each alternate key. Figure 5-1 shows the general structure of an indexed file that has been defined with only a single key. Figure 5-2 depicts an indexed file defined with two keys -- a primary key and an alternate key. This means that a file with many keys will be larger than one with few keys. More disk accesses will be required to add a record to that file because each key must be entered into each index.

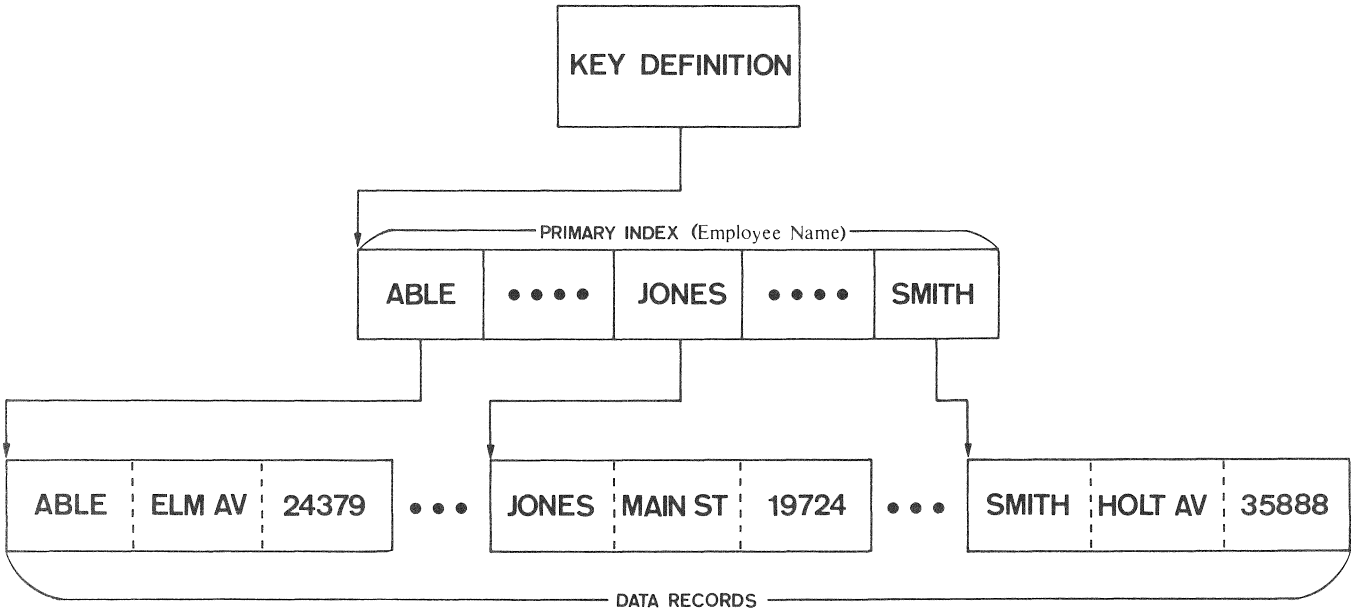

Figure 5-1 Single-Key Indexed File Organization

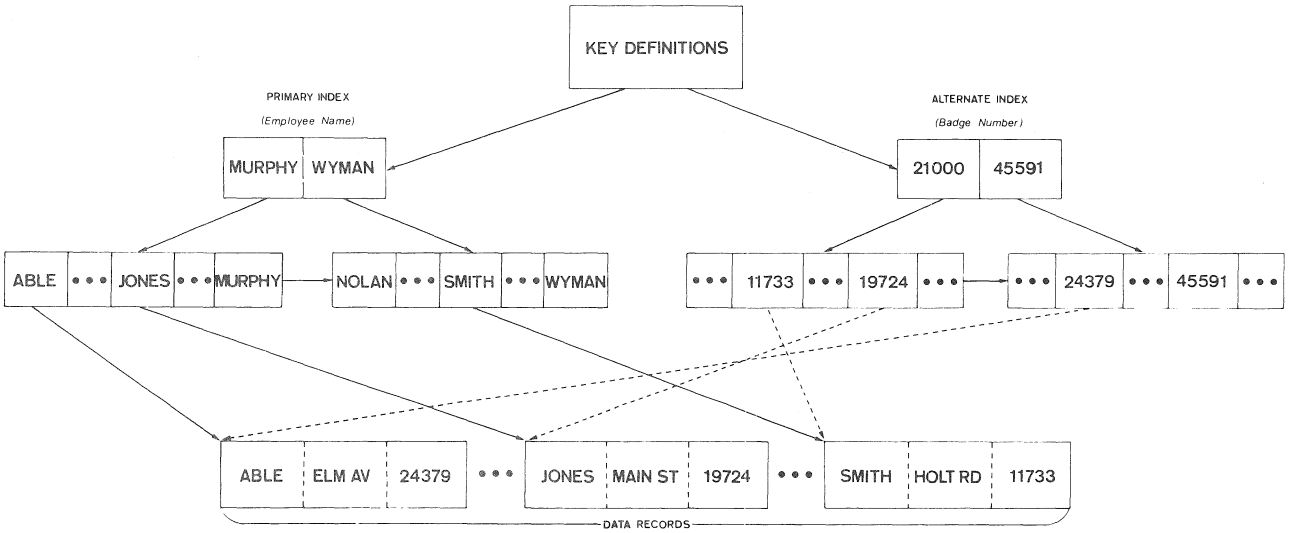

Figure 5-2 Multi-key Indexed File Organization

5.1.3.2 Defining Keys - When defining keys at file creation time, you specify two characteristics for each key:

1. Whether duplicate key values are allowed

2. Whether a key value can change (alternate keys only)

When you specify that duplicate key values are allowed, you are indicating that more than one record in the file can have the same value in a given key. Such records, therefore, have the same record identifier (for that key). The capability to allow duplicate key values further distinguishes indexed files from relative files. In relative files, the record identifier, representing a relative record number, is always unique.

A personnel file can serve as an example of the use of duplicate keys. At file creation time, the creator of the file could define the department name field as an alternate key. As programs store (STORE) records into the file, the alternate index for the department name key field will contain multiple entries for each key value (for example, PAYROLL, SALES, ADMINISTRATION) since departments are composed of more than one employee each. When such duplication occurs, RMS-11 stores the records so that they can be retrieved in first-in/first-out (FIFO) order.

Using this personnel file, an application could be written to list the names of employees in any particular department -- for example, in SALES. By reading (READ) the file by alternate value SALES, the application would obtain the first record written into the file containing this value. Then, the application could switch to sequential access (READS) and successively obtain records with the same value, SALES, in the alternate key field. Part of the logic of the application would have to determine the point at which a sequentially accessed record no longer contained the value SALES in the alternate key field. The program could then switch back to random mode (READ) and access the first record containing a different value (for example, PAYROLL) in the department name key field.

The second key characteristic (key value can change) indicates that records can be read and then written back into the file with a modified value in the key. (This is not permitted for the primary key.) When such modification occurs, RMS-11 automatically updates the appropriate index to reflect the new key value. You can specify this characteristic only for alternate keys. Further, when specifying this characteristic, you must also specify that duplicate key values are allowed.

5.2 RMS-11 ACCESS MODES

The methods of retrieving and storing records in a file are called access modes. You must choose the organization of a file at the time you create it, but you may use different access modes to process records within the file each time you use it. Also, your program can change access modes during file processing.

There are two access modes:

1. Sequential

2. Random

The DIBOL sequential access statements are READS and WRITES. The DIBOL random access statements are READ, WRITE, STORE, and DELETE. DIBOL also supports block mode access which allow access of data in entire blocks rather than individual records (see Section 5.4.4).

5.2.1 Sequential Access Mode

You can use this mode to access all types of RMS-11 files. Sequential access means that records are retrieved in the same sequence as they appear in the file. The organization of the file establishes this sequence; that is, the next record you read or write immediately follows the previous record read or written in the file. In the case of indexed files, sequential access retrieves records by increasing value of the specified key.

5.2.2 Random Access Mode

The random access mode allows the retrieval of records without regard to the location of the previous record read or written. The program, rather than the organization of the file, establishes the order in which records are processed. Successive requests can identify and access records anywhere in the file.

5.3 CREATING AN RMS-11 FILE

To create an RMS-11 sequential file with DIBOL, OPEN the file in O mode.

To create an RMS-11 relative file with DIBOL, OPEN the file in O mode and specify the O:R submode. Only fixed length records can be created, and the size of the records must be specified with the "RECSIZ" option.

To create an RMS-11 indexed file, use the RMSDEF utility or the ISMCRE external subroutine. If using RMSDEF, define file attributes by responding to the utility's requests for data and questions. The utility then stores the attributes within the newly created file. You also have the option of specifying an initial allocation quantity, and the utility allocates space at creation time; however, it does not write records into the file. The writing of the actual data contents of a file may be accomplished using RMSDEF, using a DIBOL application program on the Professional, using the RMSCNV utility or the RMSIFL utility. (The RMSCNV and RMSIFL utilities are described in **PRO/RMS-11: An Introduction.**)

An example using the RMSDEF and RMSIFL utilities is found in Section 5.6.

5.3.1 The Formats, Lengths, and Storage of Records

When creating a relative file, you must provide the maximum length specifications for the records the file will contain. The specified format then establishes how each record will physically appear in the file on a storage device. The specified length allows RMS-11 to verify that records written into the file do not exceed the length specified at file creation.

5.3.1.1 Record Formats - DIBOL supports two types of record formats:

1. Fixed-length

2. Variable-length

The choice of a record format depends on the file's organization. Table 5-1 shows the permissible combinations of file organizations and record formats.

TABLE 5-1

PERMISSIBLE COMBINATIONS OF

FILE ORGANIZATIONS AND RECORD FORMATS

File Organization	Record Format	
	Fixed-length	Variable-length
Sequential	Input Only (1)	Yes
Relative	Yes	Yes (2)
Indexed	Yes	Yes (2)

(1) DIBOL always creates variable-length sequential files.
(2) File must be created via RMSDEF, RMSDES, or the ISMCRE subroutine.

Fixed-length Record Format:

Records in the file are all equal in length; each record occupies the same amount of space in the file.

Variable-length Record Format:

Records in the file need not be equal in length. For retrieval of variable-length records, RMS-11 prefixes a count field to each record it writes. The count field describes the record length (in bytes). The count field is removed before a record is passed to the program. Variable-length records in files on disk devices have a one-word binary count field preceding the data field portion of each record. The length of the most recently accessed record is available to the DIBOL program via the RSTAT external subroutine.

5.3.1.2 Record Lengths - When defining the file, define the record length and the specific information for the record format selected. How RMS-11 uses this information depends upon the record format selected.

When selecting fixed-length records, indicate the actual length of each record in the file. This length specification becomes part of the information stored and maintained by RMS-11 for the file. Thereafter, if a program attempts to write a record whose length differs from the previously defined length, RMS-11 will reject the operation and DIBOL will return an error indicating the record was too long.

When creating a file with variable-length records, specify a maximum record length greater than zero. Subsequently, each record that a program actually writes must be less than or equal to this length or RMS-11 rejects the operation.

When creating a sequential or an indexed file with variable-length records, specify a maximum record length equal to zero. RMS-11 then neither checks nor enforces a maximum record length. (When a sequential file is created by means of a DIBOL OPEN for output [O mode] statement, the maximum record length is always set equal to zero.)

5.3.1.3 **Record Storage** - The unit of storage for RMS-11 relative and indexed files is the "bucket." A bucket can hold only complete records; it can never contain a portion of a record, and records cannot span bucket boundaries. The buckets can only be read one at a time.

You define the size of buckets in a file at the time you create the file. Buckets can consist of from 1 to 32 blocks (32 blocks is the maximum bucket size allowed on the Professional). When selecting a bucket size, consider file organization, record format, record size, and the internal information RMS-11 maintains in each bucket. Since this can be a complicated decision, refer to **PRO/RMS-11: An Introduction** before defining the bucket size.

RMS-11 uses bucket locking with relative and indexed files to ensure that a program can add, delete, or modify a record in a file without another program simultaneously accessing the same record. When a record is read or written, the bucket in which the record is contained is locked depending on mode. A READ can be done without locking. A READ in U mode locks the record until it is rewritten or it is unlocked. Indexes are also locked during the time records are being accessed.

When the program opens a relative or an indexed file with the declared intention of writing or updating records, RMS-11 locks any bucket accessed by the program. This locking prevents another program from accessing any record in the bucket until your program releases it. The lock remains in effect until your program either accesses another record, rewrites the locked record, or explicitly unlocks the locked bucket. RMS-11 then unlocks the locked bucket and locks the new one, if any. The unlocked bucket is then available for access by another concurrently executing program.

If you have large buckets and small records, you will lock more records when you lock a bucket. This could cause you to get many RECORD LOCKED (#40) errors when other programs try to access the locked bucket. Since this error is nonfatal, you can trap it, but you will suffer from poorer performance than with smaller buckets.

5.3.2 RMS-11 File Size

The size of an RMS-11 file is expressed as an integral number of virtual blocks. Each virtual block in a file is a unit of data whose size depends on the physical medium on which the file resides. For example, the size of virtual blocks in files on disk devices is 512 bytes. (Operating system convention established this size; it cannot be altered from the DIBOL environment.)

5.4 USING RMS-11 FILES

After DIBOL or the RMSDEF utility is used to create a file with a given set of attributes, the application program can access the file to store and retrieve data. When the application program accesses the file it may perform record operations that:

- Read a record. RMS-11 returns the existing record to your program.
- Write a record. RMS-11 adds the new record to the file.
- Update a record. RMS-11 modifies the contents of an existing record. The modified record replaces the old record.
- Delete a record. RMS-11 removes the existing record from the file (relative and index organization only).

The types of record operations allowed on a file depend on that file's organization.

5.4.1 Sequential File Record Operations

In the sequential file organization, your program can sequentially read existing records from the file (the READS statement). New records can be written sequentially (the WRITES statement). You cannot delete or update records.

5.4.2 Relative File Record Operations

The relative file organization permits greater flexibility in performing record operations than the sequential organization.

Your program can read existing records from the file using sequential (READS) or random (READ) access modes. Records can be written sequentially (WRITES) or randomly (WRITE). If the format of the records is variable-length, update operations using the READ(S) and WRITE statements can modify record length (though the length cannot exceed the maximum specified when the file was created).

You cannot physically delete records; you can zero a record by updating it appropriately.

5.4.3 Indexed File Record Operations

The indexed file organization provides the greatest flexibility in performing record operations. Your program can read existing records from the file in sequential (READS) or random (READ) access modes. New records can be written randomly (STORE). Your program can update records using WRITE (after a READ or READS) and delete records using DELETE (after a READ or READS). For the update operation, the primary key must not be changed.

5.4.4 Block Mode Access

Block mode access is supported by DIBOL on the Professional but is not currently documented in the **DIBOL-83 Language Reference Manual**. A file opened in block mode (using O:B, I:B, or U:B), can be accessed with a subsequent read or write in the form:

```
READ (ch,buffer,block#)

WRITE (ch,buffer,block#)
```

where:

ch	is the channel number.
buffer	is an alpha field or record where the data is to be stored.
block#	is an alpha field or record or literal specifying the relative block number to be read or written.

Multiple blocks will be transferred depending on the size of the buffer. An error message will be generated if the buffer size is not a multiple of 512.

5.4.5 Record Operations and RMS-11 MACROS

Table 5-2 cross references DIBOL language statements with RMS-11 file organization and record operations. It is provided for RMS-11 MACRO programmers who might wish to know which MACRO functions implement which DIBOL statements. If a DIBOL statement does not appear in the table, then it may not be used with RMS-11. If a statement appears but has the entry "-none-" in the column, that statement is either prohibited or not defined for the given RMS-11 file organization. The other entries indicate the RMS-11 MACRO function ($CLOSE, $DELETE, etc.) that corresponds to the DIBOL statement, and the functions (get, put, etc.) that can be performed in the given instance, or the access mode (seq, ran) involved.

TABLE 5-2

RMS-11 RECORD OPERATIONS

DIBOL Statement	File Organization		
	Sequential	Relative	Indexed
CLOSE	$CLOSE	$CLOSE	$CLOSE
DELETE	-none-	-none-	$DELETE -ran
OPEN-I	$OPEN -get	$OPEN -get	$OPEN -get
OPEN-O (:R)	$CREATE -put	$CREATE (O:R) -get, put,upd	-none-
OPEN-U	-none-	$OPEN -get, put,upd	-none-
OPEN-SI	-none-	-none-	$OPEN -get
OPEN-SU	-none-	-none-	$OPEN -get,put, upd,del
READ	-none-	$GET -ran	$GET -ran
READS	$GET -seq	$GET -seq	$GET -seq
STORE	-none-	-none-	$PUT -ran
UNLOCK	$FREE	$FREE	$FREE
WRITE	-none-	$PUT/$UPDATE -ran	$UPDATE -ran
WRITES	$PUT -seq	$PUT -seq	--none-

5.5 RMS-11 ERROR MESSAGES

RMS errors are translated, wherever possible, to errors for reporting as DIBOL run-time errors. RMS errors that do not correspond to existing DIBOL errors are handled as fatal RMS errors. Fatal RMS errors are displayed in the general format:

```
?UNEXPECTED RMS ERROR  decimal value
```

Here "decimal value" is an RMS-11 Macro (decimal) error number. For the RMS error messages, see the **PRO/RMS-11 Macro Programmers Guide.**

5.6 AN EXAMPLE: CREATING AN INDEXED FILE

NOTE

An RMS-11 sequential file or fixed-length relative file may be created simply by opening the file in O or O:R mode -- there is no need to use RMSDEF for creating these files.

This section presents the RMSDEF dialogue that creates the indexed file INFO.ISM. It is assumed that the device is the system disk (SY:) and that the account is [200,15].

The record EMPLOY, which is the relevant record in PRGRM1.DBL and PRGRM2.DBL, is defined as follows:

```
RECORD EMPLOY
     LNAME,    A10
     ID,       A6
     ADDRSS,   A20
     FILLR,    A40
```

ID, the employee identification number, will be used as the primary key. LNAME, the employee's last name, will be used as an alternate key.

The RMSDEF dialogue is generated on the RSX-11M AND RSX-11M-PLUS operating systems with the following command.

```
RUN $RMSDEF.TSK
```

The RMSDEF dialogue is generated on the VAX/VMS operating system with the following command.

```
MCR DEF
```

The RMSDEF dialogue is generated on the P/OS operating system with the following command.

```
RUN RMSDEF
```

The RMSDEF dialogue now follows. Comments are interspersed in the dialogue where appropriate. Although each user response must be terminated by pressing the RETURN key, this terminator (<CR> below) is explicitly shown only when a default is being accepted.

```
RUN $RMSDEF.TSK
DO YOU WANT TO GENERATE A COMMAND FILE FOR FUTURE USE(NO)?<CR>
```

No command file is desired.

```
ENTER FILE SPECIFICATION:INFO.ISM
```

INFO.ISM is the file specification desired for the indexed file being created.

```
IF THE FILE ALREADY EXISTS, DO YOU WANT TO SUPERSEDE IT(NO)?<CR>
```

If a file named INFO.ISM already exists, do not delete it (create a new version).

```
ENTER FILE ORGANIZATION(SEQ):IDX
```

The file is to be an indexed file. The options are SEQ (sequential), REL (relative), and IDX (indexed).

```
ENTER RECORD FORMAT(VAR):FIX
```

The only options are FIX (fixed) and VAR (variable).

The record EMPLOY is of FIXed size.

```
ENTER MAXIMUM RECORD SIZE:76
```

76 bytes (or characters) is the size of the record EMPLOY.

```
DO YOU WANT CARRIAGE RETURN CONTROL(YES)?<CR>
```

This makes the file look neat when you print it or display it on a terminal.

```
IT'S TIME TO DEFINE THE PRIMARY KEY
ENTER DATA TYPE(STR):<CR>
```

Field ID contains a string (STR) of ASCII characters. (This would be true even if ID were a decimal field rather than an alpha field.)

```
ENTER POSITION OF KEY:10
```

The field ID starts with the eleventh byte of the record. If it had started with the first byte of the record its position would have been 0.

```
ENTER SIZE OF KEY:6
```

The field ID is 6 characters long.

```
    ENTER NAME OF KEY(NONE):ID
```

The name of the primary key field is ID. The default is to assume that the key field has no name.

```
    WILL YOU ALLOW DUPLICATE KEYS(NO)?<CR>
```

Each record is to have a unique value for the primary key.

```
    DO YOU WANT TO DEFINE MORE KEYS(NO)?Y
```

The answer is Yes. The field LNAME is to be an alternate key.

```
    ENTER DATA TYPE(STR):<CR>
```

LNAME contains a string (STR) of characters.

```
    ENTER POSITION OF KEY:0
```

The field LNAME starts at the beginning of the record.

```
    ENTER SIZE OF KEY:10
```

The field LNAME is 10 characters long.

```
    ENTER NAME OF KEY(NONE):LNAME
```

The name of the first (and in this case the only) alternate key field is LNAME.

```
    WILL YOU ALLOW DUPLICATE KEYS(YES)?<CR>
```

Different employees may have the same last name.

```
    WILL YOU ALLOW KEYS TO CHANGE(YES)?<CR>
```

If an employee quits or is fired, his identification number may be given to a newly hired employee.

```
    DO YOU WISH TO DEFINE A NULL KEY VALUE(NO)?<CR>
```

You would want a null key value if there were some records that could not be accessed by this alternate key.

```
    JUST FINISHED ALTERNATE KEY NUMBER  1
    DO YOU WANT TO DEFINE MORE KEYS(NO)?<CR>
```

No additional alternate keys are desired.

```
    DO YOU WANT TO DEFINE AREAS(NO)?<CR>
```

Areas are discussed in PRO/RMS-11: An Introduction.

DO YOU WANT PLACEMENT CONTROL(NO)?<CR>

If placement control were chosen, you would be able to pick a specific location on the disk in which to place the file or an area of the file. Placement control is of no use when operating in the tool kit environment.

ENTER INITIAL ALLOCATION IN BLOCKS(0):40

Assume there are somewhat less than 200 records in the sequential file INFO.DDF that is going to be used to populate the file INFO.ISM (see the end of this section). Since each record is 76 bytes long, the initial allocation should be at least 15,200 bytes. Since 30 blocks contain 15,360 bytes, 30 blocks might seem a reasonable initial allocation. However, in the present case, it is expected that the file will probably grow by about a third of its original size. Therefore, about one-quarter of the initial allocation should be set aside for those records that will be added later. An initial allocation of 40 blocks is reasonable under these conditions. For additional discussion of data storage space requirements, see Section 5.7.

ENTER BUCKET SIZE(1):4

Note that the maximum bucket size allowed is 32. For additional discussion of bucket size see **PRO/RMS-11: An Introduction.** The value entered here will determine the size of buffers to be allocated when the file is OPENed and overrides the value specified with the PROC statement.

ENTER DEFAULT EXTENSION QUANTITY IN BLOCKS(0):4

In the present case, although the file is expected to grow by about a third of its original size, it is not expected to grow by much more than that. Therefore, since it is felt that little or no extension will be necessary beyond the initial allocation, this minimal extension quantity of 4 was chosen.

DO YOU WANT A CONTIGUOUS FILE(NO)?<CR>

If you choose to have a contiguous file, make sure your initial allocation is big enough to handle as many records as you will ever want to enter. If the file is extended it becomes noncontiguous.

THESE QUESTIONS ARE FOR THE PRIMARY KEY:

THE BUCKET SIZE IS 4 BLOCKS.
ENTER FILL NUMBER FOR DATA BUCKETS(0):1536

That is, 1536 bytes in each bucket are allotted for receiving data when the file is initially populated. Since the bucket size is 4 blocks (2048 bytes), this sets aside one quarter of the bucket (2048 - 1536 = 512 = one quarter of the bucket) for records added after the file is populated. By leaving empty space in each bucket, bucket splitting may be avoided until buckets become full. Bucket splitting is a high overhead operation that can have a large effect on system performance.

Remember that the first three digits of the ID number represent a department and the last three digits represent an employee number within that department. Therefore, with respect to the ID number, records will be more or less randomly scattered throughout the file. This fact, together with the expected growth of the file, means that new records should be adequately handled with this fill number. (Remember that a growth of one-third the original size of the file equals one-fourth the final size of the file.)

```
    ENTER FILL NUMBER FOR INDEX BUCKETS(0):1536
    THESE QUESTIONS ARE FOR ALTERNATE KEY NUMBER   1

    THE BUCKET SIZE IS 4 BLOCKS.
    ENTER FILL NUMBER FOR DATA BUCKETS(0):1536
    ENTER FILL NUMBER FOR INDEX BUCKETS(0):1536

    SPECIFY PROTECTION BY CLASS:
    OWNER (RWED ALLOWED):
    GROUP (RWED ALLOWED):
    SYSTEM (RWED ALLOWED):
    WORLD (R ALLOWED):

    YOUR FILE HAS BEEN CREATED!!! -- SY:[200,15]INFO.ISM

    ENTER FILE SPECIFICATION:^Z
```

CTRL/Z exits.

To populate the indexed file INFO.ISM with the data contained in the sequential file INFO.DDF created by PRGRM1.DBL, use RMSCNV as follows:

On RSX-11M or RSX-11M-PLUS

```
    RUN $CNV
    CNV>INFO.ISM/FO:ISM=INFO.DDF
    CNV>^Z
```

On VAX/VMS

```
    MCR CNV
    CNV>INFO.ISM/FO:ISM=INFO.DDF
    CNV>^Z
```

5.7 DATA STORAGE SPACE REQUIREMENTS

The space RMS-11 requires to store data is proportional to the organization of the file -- and the processing capabilities of that organization:

Sequential File Organization

RMS-11 adds an empty byte to the size of your data to align each record with a word boundary (a word equals two bytes). However, even if this empty byte is added, it is not included in the record byte count. When the file contains variable length records, RMS-11 adds a record-length field of two bytes to each record.

Relative File Organization

RMS-11 constructs a series of record storage cells based on the length of the records. The cells are one byte longer than the fixed size of fixed-length records or three bytes longer than the maximum size specified for variable-length records.

Indexed File Organization

RMS-11 adds to your data:

- an index for each defined key
- fifteen bytes of formatting information for each bucket
- a seven-byte header for each record
- a record-length field for each variable-length record
- other overhead of varying lengths for records RMS-11 moves during file activity and for deleted records

An additional fact to keep in mind is that records cannot cross bucket boundaries. In the example given above, the bucket size chosen is 4 blocks and the record size is 76 bytes. Therefore, a maximum of 26 records can be placed in a bucket. If there were indeed 200 records, they would not be able to fit into the 30 blocks originally suggested for the initial allocation.

For additional information see **PRO/RMS-11: An Introduction.**

CHAPTER 6

THE DIBOL DEBUGGING UTILITY (DDT)

DDT (DIBOL Debugging Technique) is a utility that allows you to interact with your DIBOL program while it is executing. The program is run with DDT after it has been compiled on either a host system or the Professional 300 system; it is run with (or without) DDT only on the Professional.

6.1 FEATURES

The features of DDT are intended to aid the programmer in locating problems; examining and modifying data values; and testing program execution directly without having to edit, compile, and rebuild the application again. Specifically, you may:

- Set predetermined stopping points.
- Examine and/or alter the contents of variables.
- Single step through lines of a DIBOL program.
- Trace through sequences of XCALL nestings.

6.2 PREPARING FOR DDT

This section reviews the procedures required to compile, link, and run with DDT.

6.2.1 Compiling

The main program, as well as all subroutines which are to be debugged, must be compiled for use with DDT by specifying the DDT option (/D) in the DIBOL compiler command. This option generates a symbol table used by DDT.

If certain subroutines are known to be already debugged, you may compile your program specifying /D only for those modules you intend to further debug.

6.2.2 Application Building

A DIBOL task must be linked with a special DDT module in order for DDT to be available at run time. See Chapter 4 in this manual for detailed information.

6.2.3 DDT Operation

6.2.3.1 Running DDT - Control is initially passed to DDT whenever a program compiled and application built for DDT operation is run. DDT outputs its version number and, on the next line, the hyphen prompt. This is illustrated below for a program, which has been compiled and linked for DDT:

```
execute program
DIBOL DDT V01.00
-
```

At this point, any valid command discussed in Section 6.3 can be entered.

6.2.3.2 Using a Terminal Connected to the Printer Port - A terminal connected to the Professional printer port can be used to monitor and enter the DDT commands and responses. In this way the normal screen display is not disrupted. To do this TT2: (the printer port) must be assigned to DDTLUN:

```
ASG=TT2:17
```

Normally the DDT logical unit number (LUN 17) is assigned to TI: (the Professional terminal) in the PAB command file (.CMD).

6.2.3.3 Failure to Properly Prepare for DDT - If you forget to perform one of the required steps in Sections 6.2, the program will exhibit the following characteristics:

- If no DDT was requested during compilation, but DBLDDT.OBJ was linked when the application was built, the program will respond to DDT except for those commands that examine and/or alter the contents of variables.

- If no DDT was requested during application build, the program will run as though no ddt were requested.

6.2.3.4 Error Messages - See the **Professional Tool Kit DIBOL Message Manual** for the DDT error messages and their meanings.

6.3 DDT COMMANDS

This section discusses valid DDT commands. In the following text, when the term routine is used it refers to a specific program module; either the main program or an external DIBOL subroutine.

For future reference, the DDT commands and command formats are listed below in the order they appear in this section.

Task	Command
Start or resume execution	CTRL/Z
Single step	CTRL/G
Setting breakpoints	$[name:]nnn
Clearing breakpoints	$[name]
Iteration of breakpoints	>n
Examining variables	vvv=
Setting variables	vvv=nnn
Extended variable manipulation	++vvv= or ++vvv=nnn
Subroutine traceback	^

DDT commands (except the CTRL/Z and CTRL/G commands) must be followed by a carriage return (<CR>).

6.3.1 Program Execution Control

Program execution control has two functions: it allows you to resume execution after a breakpoint has been encountered; and it allows you to single step through individual DIBOL statements to see if they are being properly executed.

6.3.1.1 Program Execution - To start or resume execution of the DIBOL routine from a DDT breakpoint, enter the following command in response to the DDT prompt:

```
-CTRL/Z
```

There are no arguments. The current routine simply starts or resumes execution.

6.3.1.2 Single Step - It is frequently desirable to know which branch of a computed GOTO, or of a complicated IF statement the program will take. The single step command executes the next instruction in the routine and halts. To single step, enter the following command in response to the DDT prompt:

```
-CTRL/G
```

There are no arguments. The routine executes the present instruction and returns the following message:

```
AT LINE xxxx IN ROUTINE yyyy
-
```

where:

xxxx — is the line number of the next instruction to be executed.

yyyy — is the name of the routine in which line xxxx resides.

You may now enter any DDT command in response to the DDT prompt.

Example:

Assume there is a conditional GOTO statement at line 47 in routine SUB3 and you want to find the next instruction to be executed. First, while in subroutine SUB3, set a breakpoint (see Section 6.3.2.1) at line 47:

```
$47
```

Start execution of the routine by issuing a CTRL/Z. When line 47 is reached, the display will be:

```
DDT BREAK AT LINE 47 IN ROUTINE SUB3
-
```

Respond to the prompt with CTRL/G. The display will be:

```
AT LINE _nn IN ROUTINE SUB3
-
```

where:

nn — is the next instruction to be executed.

6.3.2 Breakpoint Control

A breakpoint is a user-determined stopping point within a routine. Breakpoints are used in a routine to exercise other DDT capabilities.

6.3.2.1 Setting Breakpoints - Type the following command in response to the DDT prompt to set a breakpoint:

```
-$[name:]nnn
```

where:

name — is the name of the routine in which the breakpoint is to be set. If a breakpoint is to be set in the main program, the name of the first routine specified in the link command (by convention, the root segment) should be used. Otherwise, the name of the routine should match the name given in the subroutine statement. If the name argument is omitted, the current routine is assumed.

nnn — is the line number at which the routine is to halt.

- The line at which the routine is halted has not yet been executed.
- A maximum of eight breakpoints may be set at any one time.
- Only one breakpoint is allowed in any main program or subroutine at any given time.
- A breakpoint in the data section has no meaning and will never cause a break.

Example:

```
-$SUB1:50
```

sets a breakpoint at line 50 in subroutine SUB1.

```
-$21
```

sets a breakpoint at line 21 in the current routine.

6.3.2.2 Clearing Breakpoints - Previously set breakpoints may be cleared by typing the following command in response to the DDT prompt:

```
-$[name]
```

where:

name — is the name of the routine in which the breakpoint is to be deleted. If name is omitted, the breakpoint in the current routine is cleared.

- The breakpoint in a routine need not be deleted before a breakpoint is set at a new line in that routine.

- Setting a new breakpoint automatically deletes any other breakpoint in that routine.

Example:

```
-$SUB2
```

clears the breakpoint in subroutine SUB2.

```
-$56
```

sets a breakpoint at line 56 in the current routine and clears any prior breakpoint in that routine.

6.3.2.3 Iteration of Breakpoints - To test the effects resulting from iterative procedures, it is sometimes useful to set a breakpoint in a loop and pass through it several times before allowing execution to halt. This is accomplished with the following command in response to the DDT prompt:

```
->n
```

where:

>	is the iteration specifier.
n	is the iteration count. This is the number of times the breakpoint is to be encountered before execution is halted.

- The iteration count can be set only in the current routine.

- You must be at the breakpoint before issuing the iteration command.

- Execution is halted the nth time the breakpoint is encountered.

Example:

Assuming that a breakpoint is set in a loop at line 25 of the current routine and the program executes until reaching this point, the response will be:

```
DDT BREAK AT LINE 25 IN ROUTINE XXX
-
```

where:

XXX is the name of the current routine.

You might respond with an iteration count and execution command:

```
->8
-CTRL/Z
```

The routine then loops through this location; stopping the eighth time it reaches line 25. The response is:

```
DDT BREAK AT LINE 25 IN ROUTINE XXX
-
```

6.3.3 Variable Manipulation

Variable manipulation allows you to change or examine variables in a routine to determine whether or not they are being correctly handled.

6.3.3.1 Setting Variables - Variables may be set (loaded) with any desired value by using the following command in response to the DDT prompt:

```
-vvv=nnn
```

where:

vvv is the variable name.

nnn is the value you want to assign to the variable.

- If the length of nnn is too long to store in vvv, the data is left justified in the field and the excess right-hand characters are truncated. This is true for both alpha and decimal fields.
- Do not use single quotes when specifying alpha data.
- A field, alpha or decimal, can be cleared by entering a space for an assigned value.
- Decimal literal subscripts, single or double, can also be used to set variables.

Examples:

```
-VAR1=ABCD
```

Assigns the value of ABCD to VAR1.

```
-VAR1='ABCD'
```

Assigns the value of 'ABC to VAR1.

6.3.3.2 Examining Variables - Variables may be examined to verify their contents with the following command in response to the DDT prompt:

```
-vvv=
```

where:

vvv — is the variable name. Decimal literal subscripts,either single or double, may be used with the variable name to access an array element, a part of a field, or data in an unlabeled field.

Example:

Assume you have stopped at a breakpoint; then:

```
-VAR1=
```

results in a display of the present contents of this variable.

6.3.3.3 Extended Variable Manipulation - It is possible under the specific circumstances explained here to examine, or to set, a variable used outside the current routine. This may be done only when the variable is defined in the routine which called the current routine or is defined in one of the routines in the chain of calls which led to the current routine. For example:

```
-++VAR2=
```

will return the current value of VAR2 located in the chain of routines which called the current routine. The two plus signs indicate that the variable was defined in a routine located two calls back (two levels of nesting) in the chain which led to the current routine. Also:

```
-++VAR2=EFGH
```

will set VAR2 to the value EFGH.

6.3.4 Subroutine Traceback

The subroutine traceback feature allows you to determine whether or not the calling sequences (XCALL statements) are executing in the expected manner. The output is a list of the routines and the line numbers in those routines of all the related preceding XCALL statements back to the main program. To obtain this list, enter the following command (a caret, up arrow, or circumflex) in response to the DDT prompt:

```
_^
```

There are no arguments. The circumflex (^) causes the list to be generated.

Example:

Assume you have halted in a subroutine at a DDT breakpoint, or you have single stepped to the current position, and you need to know how you arrived at this point from the main program. The command and traceback list might look like the following:

```
_^
AT LINE 37 IN ROUTINE SUB3      (current location)
AT LINE 192 IN ROUTINE SUB2     (SUB3 called from SUB2, line 192)
AT LINE 21 IN ROUTINE MAIN      (SUB2 called from MAIN, line 21)
-
```

You are still in routine SUB3 and may enter any DDT command.

CHAPTER 7

DIBOL INTERFACE TO FMS

This chapter assumes that you are familiar with FMS. The DIBOL subroutines explained in this chapter provide a way for you to use the XCALL statement to incorporate FMS functionality in your DIBOL programs; each subroutine calls a function provided by FMS. This chapter presents argument data types, calling syntax, and task building information for the DIBOL interface to FMS-11.

For more information on FMS-11, see the list of related documents in the preface.

In DIBOL applications, all values passed to and from the form driver must be ASCII string variables or literals. When the form driver returns a string value, the length is the length of the field, including any trailing spaces or fill characters. String values that are shorter than the DIBOL variables to which they are assigned are left-justified and the fields are blank filled. String values that are longer than the variables to which they are assigned result in the DIBOL run-time error message #31 (Argument wrong Size) and cause the form driver to set the status code to -22.

7.1 DIBOL DATA TYPES FOR FORM DRIVER ARGUMENTS

All the forms driver arguments used in DIBOL XCALLs to FMS have a common meaning. These are explained in Table 7-1.

TABLE 7-1

ARGUMENT DATA TYPES

Argument Abbreviation	Explanation (purpose, data type, and data structure)
chan	Channel number: ASCII numeric string variable or literal
fid	Field name: six-byte ASCII string variable or literal
fidx	Field and named data index: ASCII numeric string variable or literal
flen	Field length: ASCII numeric variable or literal
flnm	Form library file specification: ASCII string variable or literal (must incorporate a trailing space)
fnam	Form name: six-byte ASCII string variable or literal

Continued on Page 7-2

fval	Named data value, one or more field values, text for display on the bottom screen line: ASCII string variable or literal (the size depends on the application)
tid	Keyboard string that identifies the terminal. The default can be identified by KBn: where n is the terminal number.
impure	Impure area: byte array (using the impure area size that the Form Editor and the Form Utility report, the size of the array should be 64 bytes larger than the largest impure area for the forms that the application uses).
line	First line for a displayed form: ASCII numeric variable or literal
size	The size of the impure area in bytes: ASCII numeric value
status	Call completion status: ASCII numeric string variable
stat2	RSTS/E system error code: ASCII numeric string variable
term	Field terminator code: ASCII string variable or literal

7.2 SYNTAX FOR THE CALLS

All Form Driver calls use the XCALL statement. The following table summarizes the principal purposes and shows the full XCALL statement syntax for each call. The arguments that you must supply are in lower case letters, and optional arguments are enclosed in square brackets ([and]). The formats of calls that have no arguments are listed separately. The argument abbreviations and purposes are fully described in Table 7-2.

TABLE 7-2

DIBOL FORM DRIVER CALLS

Call Abbreviation	Summary and Forms
CLRSH	Clears the entire screen and displays the form with the default field values. If a line number is specified, it is used as the first line of the form. XCALL CLRSH (fnam[,line])

Continued on Page 7-3

FGCF	Returns the field name from the Form Driver argument list (and if it is an indexed field, its index). XCALL FGCF (fid(,fidx])
FGET	If a field name is specified, FGET gets and returns the value for the field and the field terminator used. If no field name is specified, FGET places the cursor at the lower right corner of the screen and deactivates all operator responses except the RETURN and ENTER keys. XCALL FGET ([fval,term,fid[,fidx]])
GETAF	Gets and returns the value, field name (and if it is an indexed field, its index), and the field terminator used for the field that the operator chooses. XCALL GETAF (fval,term,fid[,fidx])
GETAL	If the call includes an argument, GETAL gets and returns a concatenated string of all field values (and optionally the last field terminator used). If no arguments are specified, GETAL gets all values from the operator but only stores them in the impure area. XCALL GETAL ([fval[,term]])
IDATA	Gets and returns the named data value that has the specified index. XCALL IDATA (fidx,fval)
FINIT	Supplies to the Form Driver the name and size of the impure area to use. The 'size' argument isn't needed unless the 'status' argument is being passed. Size is conveyed by the DIBOL 'size, origin descriptor' passed for every argument. XCALL FINIT (impure[,size[,status]])
FINLN	Gets and returns a concatenated string of the field values for the current line of the scrolled area that contains the specified field name and the last terminator used. XCALL FINLN (fid,fval,term)
LCHAN	Supplies the Form Driver with the I/O channel (LUN) to use for reading a form library file. XCALL LCHAN (chan)

LCLOS	Closes the current form library file. XCALL LCLOS
FLEN	Returns the length of the specified field. XCALL FLEN (flen,fid[,fidx])
LOPEN	Opens the specified form library file. XCALL LOPEN (flnm)
NDATA	Gets and returns the named data value that has the specified named data label. XCALL NDATA (fid,fval)
OUTLN	Displays the specified string of field values in the current line of the scrolled area that contains the specified field. XCALL OUTLN (fid,fval)
FPFT	If the call includes an argument, FPFT processes the specified field terminator and identifies the appropriate field as the current field. To get the name of the field, use the FGCF call. If the specified terminator is a scrolled area terminator, the name of a field in the intended scrolled area must be specified; if a string of values is also specified, they will be displayed on the top or bottom line of the scrolled area after the terminator is processed. If no argument is included, FPFT processes the last terminator that was used. XCALL FPFT ([term[,fid[,fval]]])
FPUT	Displays the specified value in the specified field. XCALL FPUT (fval,fid[,fidx])
PUTAL	Displays values in all fields of the form. If a concatenated string of values is supplied, each value must be the same length as the field in which it is to be displayed. The values must be in the same order that the GETAL call would produce for the form. Values from the supplied string are displayed in the first fields of the form; defaults are displayed in any fields that remain. If no string of values is supplied, default values are displayed in all fields. XCALL PUTAL ([fval])

FPUTL	If an argument is specified, FPUTL displays the specified string on the bottom line of the screen. If no argument is specified, FPUTL clears the bottom line. XCALL FPUTL ([fval])
RETAL	Returns the current values for all fields in the form in the same order that the GETAL call would produce. XCALL RETAL (fval)
FRETN	Returns the current value of the specified field. XCALL FRETN (fval,fid[,fidx])
FSHOW	Clears the area of the screen specified when the form was created and displays the form with default field values. If a line number is specified, FSHOW uses it as the first line for the form. XCALL FSHOW (fnam[,line])
SPOFF	Turns off the supervisor-only mode and allows the operator to enter and change data in fields to which the supervisor-only attribute was assigned with the Form Editor. XCALL SPOFF
FSPON	Turns on the supervisor-only mode and prevents the operator from entering or changing data in fields to which the supervisor-only attribute was assigned with the Form Editor. XCALL FSPON
FSTAT	Returns the status code for the last call that was processed as the value of the first argument. The value of the second argument is meaningful as an RMS system error code (depending on the version of the Form Driver in use) only if the value of the first argument is -4 or -18. These two arguments indicate an error occurred while trying to open or read a form library file. XCALL FSTAT (status[,stat2])

7.3 BUILDING A DIBOL TASK

The following example shows how to build a nonoverlaid FMS DIBOL application. The command file builds with the Form Driver object library. In order to include this object library when application building, the following three steps should be added.

STEP 1

Compile the program using the /B switch causing the DIBOL compiler to create default .CMD and .ODL files for you.

STEP 2

Edit the .ODL file and add '-LB:[1,5]FDV/LB' to the end of the line beginning with 'DBLIB$:' . (See original and modified files, on the following page.)

ORIGINAL FMSTEST.ODL

```
                .ROOT     APOBJ$-DBLIB$-RMS$
APOBJ$:         .FCTR     program
DBLIB$:         .FCTR     LB:[1,5]DBPLIB/LB-LB:[1,5]DBPOSL/LB
RMS$:           .FCTR     RMSROT
@LB:[1,5]RMSRLX.ODL
                .END
```

MODIFIED FMSTEST.ODL

```
                .ROOT     APOBJ$-DBLIB$-FMS$-RMS$
APOBJ$:         .FCTR     program
DBLIB$:         .FCTR     LB:[1,5]DBPLIB/LB-LB:[1,5]DBPOSL/LB
FMS$:           .FCTR     LB:[1,5]fmslib/LB
RMS$:           .FCTR     RMSROT
@LB:[1,5]RMSRLX.ODL
                .END
```

where:

fmslib may be either:

- FDVDBG, which includes error messages that are helpful during debugging.
- FDV, which is a smaller module without the messages.

STEP 3

Build your program with PAB.

CHAPTER 8

DIBOL INTERFACE TO P/OS SYSTEM SERVICES, CALLABLE IMAGES, AND ROUTINES

This chapter is intended to be used in conjunction with the **Professional Tool Kit User's Guide** and the P/OS System Reference Manual.

Only those services, images, and routines that must be interfaced from DIBOL are documented. The first four sections (Sections 8.1 through 8.4) include the menu, message, help, and miscellaneous services available through POSRES. Also included (Section 8.5) are the calls to the SORT and PROSE utilities. Printing from a DIBOL application is accomplished via the DIBOL PRINT subroutine rather than the P/OS PRINT utility. The final section (Section 8.6) covers callable system routines. These three groups are documented in Chapters 5 and 6 of the **Tool Kit User's Guide** and as Chapter 8 of the **P/OS System Reference Manual**, respectively. The sections of this chapter parallel the contents of those chapters.

This chapter shows the proper format for calls from a DIBOL program. Within each of these categories an explanation of the arguments (purpose, data type, and data structure) precedes the presentation of the calls and the calling format. The explanation for the system calls is not included here. This chapter includes only the DIBOL parameters. It should be consulted in conjunction with the Professional Tool Kit documentation where the Professional system calls are fully documented.

The DIBOL interface to FMS is documented in Chapter 7 of this manual. The DIBOL interface to CORE Graphics is documented in Chapter 9 of this manual.

NOTE

> Most of the system calls require specification of both the variable name and the size. The DIBOL SIZE external subroutine may be used to obtain the size of the variable.

General Information for Sections 8.1 through 8.4

The P/OS user interface services library (POSRES) offers service routines for displaying menus, on-line help, and messages from frames created with FDT or stored data. These displays comprise the user interface for a P/OS application. These interface routines can be called from high-level languages. These sections describe how service routines can be called in different combinations from DIBOL to display user interface frames.

The reason for some menu calls having two entry points from DIBOL is because the parameters passed do not always require the same conversion.

8.1 MENU SERVICE ROUTINES

This section describes the service routines for creating, retrieving, and displaying single and multiple-choice menus.

8.1.1 Open Menu File

This routine opens the specified menu definition file. Only one menu file can be open at any time. Any open menu file is closed before the requested file is opened. After opening a menu file, one frame at a time can be read from it.

Call

```
XCall DMFIL (status,filename,maxlen)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

filename — An alpha expression specifying the menu definition file name and type. The system searches the directory in which the application resides unless a directory spec is included.

maxlen — A decimal expression indicating the length of the filename parameter.

Example

```
RECORD
    STATUS     ,2D3
PROC
    XCALL DMFIL ( STATUS, 'BASETEST.MNU', 12 )
    IF (STATUS(1).LT.1) GOTO ERROR
```

8.1.2 Read Menu Frame

This routine reads the specified menu frame into the static menu buffer.

Call

```
XCall DMFRA (status,frameid,maxlen[,globalstr,maxlen,length])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

frameid — An alpha expression containing the frame identifier of the desired frame.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

globalstr — An optional alpha variable of any length. This parameter is used to pass back the global action string associated with the menu frame, if present.

length — A decimal variable which is returned indicating the number of characters in globalstr that were actually passed back.

Example

```
RECORD
    GLOACT     ,A80
    LENGTH     ,D2
    STATUS     ,2D3
PROC
    XCALL DMFRA ( STATUS, 'FRAME001', 8, GLOACT, 80, LENGTH )
    IF (STATUS(1).LT.1) GOTO ERROR
```

8.1.3 Show Single-Choice Menu

This routine displays a frame from the default static menu buffer.

Call

```
XCall DMEN (status,action,maxlen,length,display,addopt,
            [msg1,maxlen,msg2,maxlen])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

action — An alpha variable that will receive the action string for the selected option.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

length — A decimal variable which is returned indicating the number of characters in action that were actually passed back.

display — Reserved for future use. Pass a literal zero.

addopt — A decimal expression indicating whether to show an Additional Options flag on the screen.

0 = don't show flag, non-0 = show flag

msg1/2 — An alpha expression containing text to be displayed at the bottom of the menu.

Example

```
RECORD
    ACTION     ,A80
    LENGTH     ,D2
    MSG1       ,A30 ,'THIS IS THE FIRST MESSAGE LINE'
    MSG2       ,A31 ,'THIS IS THE SECOND MESSAGE LINE'
    ADDOPT     ,D1  ,0
    STATUS     ,2D3
PROC
    XCALL DMEN ( STATUS, ACTION, 80, LENGTH, 0, ADDOPT,
&                Msg1, 30, Msg2, 31 )
    IF (STATUS(1).LT.1) GOTO ERROR
```

8.1.4 Unpack Menu Buffer

This routine unpacks the static buffer into fields.

Call

```
    XCall DMUNP (status,fieldid,maxlen,buff,maxlen,length
                 [,fieldid,maxlen,buff,maxlen,length ...])

    XCall DMUN1 (status,'DFLT',maxlen,defopt,
                   'KEYWnn',maxlen,offset,keylen
                   [,'KEYWnn',maxlen,offset,keylen ...])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

fieldid — An alpha expression specifying what field is to be unpacked.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

buff — An alpha variable used to pass the value of the fieldid. For example, if fieldid is 'TITL' then buff receives the title text.

length — A decimal variable which is returned value indicating the number of characters that were actually placed in the buffer.

defopt — A decimal variable that is returned the default option number.

offset — A decimal variable receiving the number of characters from the start of the option at which the keyword begins.

keylen — A decimal variable receiving the number of characters in the keyword.

Example

```
RECORD
    TITLE      ,A80       ; Will receive the 'TITL' field
    PROMPT     ,A80       ;  "      "     "  'PRMT'   "
    GBLHLP     ,A8        ;  "      "     "  'GHLP'   "
    TXT        ,3A80      ;  "      "     "  'TEXTNN' Fields
    OPT        ,12A80     ;  "      "     "  'OPTNNN' Fields
    ACT        ,12A80     ;  "      "     "  'ACTNNN' Fields
    HLP        ,12A8      ;  "      "     "  'OHLPNN' Fields
    DEFLT      ,D2        ;  "      "     "  'DFLT' Field
    KEY        ,12D2      ;  "      "     "  'KEYWNN' key posn
    LEN        ,12D2      ;  "      "     "  'KEYWNN' key length
    LTXT       ,3D2       ;  "      "     "  Length of TXT(X)
    LOPT       ,12D2      ;  "      "     "  Length of OPT(X)
    LACT       ,12D2      ;  "      "     "  Length of ACT(X)
    LHLP       ,12D1      ;  "      "     "  Length of HLP(X)
    LGHELP     ,D1        ;  "      "     "  Length of GBLHLP
    LTITLE     ,D2        ;  "      "     "  Length of TITLE
    LPRMPT     ,D2        ;  "      "     "  Length of PROMPT
    STATUS     ,2D3       ;  "      "     "  STATUS of the call
PROC
                          ; Unpack title, global help, and prompt fields.
    XCALL DMUNP ( STATUS, 'GHLP',   4, GBLHLP, 8,  LGHELP,
&                         'TITL',   4, TITLE,  80, LTITLE,
&                         'PRMT',   4, PROMPT, 80, LPRMPT)
                          ; Unpack 2 options, Help frames, Action Strings
    XCALL DMUNP ( STATUS, 'OPTN01', 6, OPT(1), 80, LOPT(1),
&                         'OPTN02', 6, OPT(2), 80, LOPT(2),
&                         'ACTN01', 6, ACT(1), 80, LACT(1),
&                         'ACTN02', 6, ACT(2), 80, LACT(2),
&                         'OHLP01', 6, HLP(1), 8,  LHLP(1),
&                         'OHLP02', 6, HLP(2), 8,  LHLP(2))
                          ; Unpack key word and default option
    XCALL DMUN1 ( STATUS, 'KEYW01', 6, KEY(1), LEN(1),
&                         'KEYW02', 6, KEY(2), LEN(2),
&                         'DFLT',   4, DEFLT )
```

8.1.5 Pack Dynamic Single-Choice Menu

This routine packs the dynamic menu buffer using specified fields.

Call

```
XCall DDPAC (status,fieldid,maxlen,buff,maxlen
             [,fieldid,maxlen,buff,maxlen...],
             ['DFLTnn',maxlen],
             ['CLRB',maxlen])

XCall DDPAl (status,'KEYWnn',maxlen,offset,length
             [,'KEYWnn',maxlen,offset,length...])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

fieldid — An alpha expression specifying what field is to be packed.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

buff — An alpha variable used to pass the value of the fieldid. For example, if fieldid is 'TITL' then buff contains the title text.

offset — A decimal expression representing the number of characters from the start of the option at which the keyword begins.

length — A decimal expression containing the number of characters in the keyword.

Example

```
RECORD
    TITLE      ,A25 ,'Title for DYNAMIC menu   '
    PROMPT     ,A30 ,'Select option and press DO    '
    TXT1       ,A30 ,'This is the first text line   '
    TXT2       ,A30 ,'And this is the second line   '
    TXT3       ,A30 ,'And finally the third one     '
    OPT1       ,A30 ,'Option 1 - the first choice   '
    OPT2       ,A30 ,'Second option - the default   '
    ACT        ,2A8 ,'Action 1','Action 2'
    HLP        ,2A8 ,'HELP0001','HELP0002'
    GBLHLP     ,A8  ,'HELP0000'
    KEY        ,2D1 ,  0 ,  0
    LEN        ,2D1 ,  8 ,  6
    STATUS     ,2D3
PROC
                    ; Clear the buffer, pack the title, text, global
                    ; help, prompt and default option.
    XCALL DDPAC ( STATUS,     'CLRB',   4,
&                             'DFLT02', 6,
&                             'GHLP',   4, GBLHLP, 8,
&                             'TITL',   4, TITLE,  25,
&                             'PRMT',   4, PROMPT, 30,
&                             'TEXT01', 6, TXT1,   30,
&                             'TEXT02', 6, TXT2,   30,
&                             'TEXT03', 6, TXT3,   30 )
                    ; Pack 2 options, Help frames and Action Strings
    XCALL DDPAC ( STATUS,     'OPTN01', 6, OPT1,  30,
&                             'OPTN02', 6, OPT2,  30,
&                             'ACTN01', 6, ACT(1), 8,
&                             'ACTN02', 6, ACT(2), 8,
&                             'OHLP01', 6, HLP(1), 8,
&                             'OHLP02', 6, HLP(2), 8,
                    ; Pack key word information. (Note: requires DDPA1
                    ; entry)
    XCALL DDPA1 ( STATUS,     'KEYW01', 6, KEY(1), LEN(1),
&                             'KEYW02', 6, KEY(2), LEN(2))
```

8.1.6 Display Dynamic Menu

This routine displays a single choice menu constructed from the dynamic buffer. The operation of a menu created with DMENU is identical to the operation of a menu displayed with the MENU routine.

Call

```
XCall DDMEN (status,action,maxlen,length,display,addopt,
             [msg1,maxlen,msg2,maxlen])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

action — An alpha variable that will receive the action string for the selected option.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

length — A decimal variable which is returned indicating the number of characters in action that were actually passed back.

display — Reserved for future use. Pass a literal zero.

addopt — A decimal expression indicating whether to show an Additional Options flag on the screen.

0 = don't show flag, non-0 = show flag

msg1,msg2 — An alpha expression containing text to be displayed at the bottom of the menu.

Example

```
RECORD
    ACTION     ,A80
    LENGTH     ,D2
    MSG1       ,A42 ,'The first message line of the Dynamic Menu'
    MSG2       ,A43 ,'The second message line of the Dynamic Menu'
    ADDOPT     ,D1  ,0
    STATUS     ,2D3
PROC
    XCALL DDMEN ( STATUS, ACTION, 80, LENGTH, 0, ADDOPT,
&                 MSG1, 42, MSG2, 43 )
    IF (STATUS(1) .LT. 1) GOTO ERROR
```

8.1.7 Pack Multiple-Choice Menu

This routine packs the multibuffer with peripheral parts of the multiple-choice menu. Peripheral parts of the menu include the title, text lines, and prompt, as well as the global action string and help frameid.

Call

```
XCall DMPAC (status,fieldid,maxlen,buff,maxlen
             [,fieldid,maxlen,buff,maxlen...],
             ['CLRB',maxlen])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

fieldid — An alpha expression specifying what field is to be packed.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

buff — An alpha variable used to pass the value of the fieldid. For example, if fieldid is 'TITL' then buff contains the title text.

Example

```
RECORD
    TITLE     ,A40 ,'Title for DYNAMIC Multiple-Choice menu  '
    PROMPT    ,A30 ,'Select option and press DO    '
    TXT1      ,A30 ,'This is the first text line   '
    TXT2      ,A30 ,'And this is the second line   '
    TXT3      ,A30 ,'And finally the third one     '
    GBLHLP    ,A8  ,'HELP0000'
    STATUS    ,2D3
PROC
              ; Clear the buffer, pack the title, text, global
              ; help, and prompt fields. Use only one XCall to be
              ; more efficient
    XCALL DMPAC ( STATUS,    'CLRB',   4,
&                            'GHLP',   4, GBLHLP, 8,
&                            'TITL',   4, TITLE,  25,
&                            'PRMT',   4, PROMPT, 30,
&                            'TEXT01', 6, TXT1,   30,
&                            'TEXT02', 6, TXT2,   30,
&                            'TEXT03', 6, TXT3,   30 )
```

8.1.8 Display Multiple-Choice Menu Frame

This routine displays a multiple-choice menu created from a combination of the multibuffer, the option-text, and of message specified in this routine.

Call

```
XCall DMMEN (status,optstr,optlen,optcnt,limit,
             numrsp,rsparr,addopt,
             [msg1,maxlen,msg2,maxlen])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

optstr — An alpha array variable containing all of the options for this menu. Each option may be up to 72 characters long.

optlen — A decimal expression indicating the length per entry in the optstr array.

optcnt — A decimal expression indicating the number of entries in the optstr array.

limit — A decimal expression specifying the maximum number of choices that the user can select.

numrsp — A decimal variable that will be set to the number of responses that the user selected.

rsparr — A decimal array variable that the option numbers that the user has selected will be placed. The number of valid entries in this array is in the numrsp argument.

addopt — A decimal expression indicating whether to show an Additional Options flag on the screen.

0 = don't show flag, non-0 = show flag

msg1/2 — An alpha expression containing text to be displayed at the bottom of the menu.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

Example

```
RECORD
    OPTSTR     ,6A5 ,'Red  ','Blue ','Green',
&                   ,'Brown','Black','White'
    NUMRSP     ,D1
    RSPARR     ,5D1
    MSG1       ,A30 ,'This is the first message line'
    MSG2       ,A31 ,'This is the second message line'
    ADDOPT     ,D1  ,0
    STATUS     ,2D3
PROC
                    ; Display the dynamic multiple-choice menu allowing
                ; them to make 2 choices.
    XCALL DMMEN ( STATUS, OPTSTR, 5, 6, 2, NUMRSP, RSPARR,
&                 ADDOPT, MSG1, 30, MSG2, 31 )
    IF (STATUS(1) .LT. 1) GOTO ERROR
                    ; The following code shows how to display the text of
                    ; the choices made
    OPEN (1,O,'TT:')
LP,
    IF (NUMRSP.EQ.0) GOTO END
    DISPLAY   (1,13,10,'CHOOSE - ',OPTSTR(RSPARR(NUMRSP)))
    NUMRSP = NUMRSP - 1
    GOTO LP
END,
    CLOSE 1
```

8.1.9 Close Menu File

This routine closes an open menu file.

Call

```
    XCall DMCLO (status)
```

status	A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the Tool Kit User's Guide for status values.

Example

```
RECORD
    STATUS     ,2D3
PROC
                        ; Assume a menu file is open at this point
                        ;
                        ; Close the menu file
    XCALL DMCLO (STATUS)
    IF (STATUS .LT. 1) GOTO ERROR
```

8.2 HELP SERVICE ROUTINES

This section describes service routines for retrieving and displaying help-text frames and help menus.

8.2.1 Open Help File

This routine opens the specified help definition file. Only one help file can be open at any time. If another help file is open, it is closed before the requested file is open. This call does not result in a display.

Call

```
XCall DHFIL (status,filename,maxlen,frameid,maxlen)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the Tool Kit User's Guide for status values.

filename — An alpha expression specifying the help definition file name and type.

maxlen — A decimal expression indicating the length of the preceding alpha expression.

frameid — An alpha expression containing the frameid for the default help frame.

Example

```
RECORD
    STATUS     ,2D3
PROC
    XCALL DHFIL ( STATUS, 'BASETEST.HLP', 12, 'HELP0000', 8 )
    IF         (STATUS(1) .LT. 1) GOTO ERROR
```

8.2.2 Specify Help Frame

This routine specifies the frame ID of the frame to be displayed when help is requested. This call does not result in a display; it simply defines the default frame.

Call

```
XCall DHFRA (status,frameid,maxlen)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the Tool Kit User's Guide for status values.

frameid An alpha expression containing the frame identifier of the desired frame.

maxlen A decimal expression specifying the length of the frameid variable.

Example

```
RECORD
    STATUS     ,2D3
PROC
    XCALL DHFRA ( STATUS, 'HELP0001', 8)
    IF (STATUS(1) .LT. 1) GOTO ERROR
```

8.2.3 Display Help Frame

This routine displays the default help frame.

Call

```
XCall DHELP (status [,frameid,maxlen] )
```

status A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

frameid An optional alpha expression containing the frame identifier of the desired frame.

maxlen A decimal expression specifying the length of the frameid variable.

Example

```
RECORD
    STATUS     ,2D3
PROC
    XCALL DHELP ( STATUS )
    IF (STATUS(1) .LT. 1) GOTO ERROR
                              ; Display a new help frame
    XCALL DHELP ( STATUS, 'HELP0002', 8 )
    IF (STATUS(1) .LT. 1) GOTO ERROR
```

8.2.4 Close Help File

This routine closes the help definition file.

Call

```
XCall DHCLO (status)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

Example

```
Record
    STATUS    ,2D3
PROC
                        ; Assume a help file is open at this point
                        ;
                        ; Close the help file
    XCALL DHCLO (STATUS)
    IF (STATUS .LT. 1) GOTO Error
```

8.3 MESSAGE SERVICE ROUTINE

This section describes the service routine for retrieving and displaying a message.

Read Message

This routine reads the specified message from the specified message file into a buffer supplied by the calling task.

Call

```
XCall DRDMS (status,filename,maxlen,frameid,maxlen,
              buffer,maxlen,length)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

filename — An alpha expression specifying the message file name and type. The system searches the directory in which the application resides unless a directory spec is included.

maxlen — A decimal expression indicating the length of the preceding alpha expression.

frameid — An alpha expression specifying the message frame identifier for the desired message.

buffer — An alpha variable in which to read the message specified by frameid.

length — A decimal variable which receives the length of the message read into the buffer.

Example

```
RECORD
    BUFFER    ,A80
    LENGTH    ,D2
    STATUS    ,2D3
PROC
    XCALL DRDMS (STATUS, 'BaseTest.MSG', 12, 'Mess0001', 8,
&                BUFFER, 80, LENGTH )
    IF (STATUS(1) .LT. 1) GOTO Error
```

8.4 MISCELLANEOUS SERVICES

Miscellaneous services includes the following routines:

- Display a fatal error message.
- Get a keystroke from the terminal.
- Parse a string.
- Display P/OS forms to name a new file and get an old file name.
- Wait for the RESUME key.
- Send a message to the Message/Status Display.

8.4.1 Fatal Error

This routine provides processing for unexpected and disabling error conditions which may prevent an application from continuing.

Call

XCall DFATL (message, msglen)

message — An alpha expression containing the message to be displayed on the bottom line.

msglen — A decimal expression containing the length of the message parameter.

Example

```
PROC
    ONERROR   ERR
ERR,
    XCALL DFATL ('A Fatal Error Has Occured',25)
```

8.4.2 Get Keystroke

This routine gets a single key stroke from the terminal. The keystroke is not echoed on the screen.

Call

```
    XCall DGETK (status)
```

status — A decimal array with two elements containing a code corresponding to which key was pressed.

In the first element of the status array, +1 indicates that the key is a single ASCII character. The second element of the array contains the ASCII code of the key.

In the first element of the status array, +2 indicates that the key is a function key. The second element of the array contains a code corresponding to the function key. See Section 5.9 of the **Tool Kit User's Guide** for function key codes.

In the first element of the status array, a value less than zero indicates that an error occurred. See Section 5.9 of the **Tool Kit User's Guide** for status values.

Example

```
RECORD
    STATUS    ,2D3
PROC
    XCALL DGETK (STATUS)
    IF (STATUS(1) .LT. 1) GOTO ERROR
    GOTO (ASCII,FUNCT), STATUS(1)
    GOTO ERROR
```

8.4.3 Parse String

This routine parses a string for a CSI sequence. The characters in the buff parameter are scanned from the left until a CSI character is found. The sequence following the CSI character is parsed and translated into a value indicating what function key terminates the string. The length of the string up to the CSI character is returned in the length parameter.

Call

```
XCall DPRSC (status,buff,maxlen,length)
```

status — A decimal array with two elements. In the first element of the array, +2 indicates that a valid CSI sequence was found. The second element of the array contains the CSI sequence code. See Section 5.9 of the **Tool Kit User's Guide** for function key values.

In the first element of the status array, a value less than zero indicates that an error occured. See Section 5.9 of the **Tool Kit User's Guide** for status values.

buff — An alpha variable specifying the buffer to be parsed.

maxlen — A decimal expression specifying the length of the buffer.

length — A decimal variable to receive the number of characters before the CSI character.

Example

```
RECORD
    STATUS     ,2D3
    BUFFER     ,A80
    LENGTH     ,D2
PROC
    OPEN (1,I,'TT:')
    READS (1,BUFFER)      ; Let the user fill in the buffer
    XCALL DPRSC (STATUS,BUFFER,80,LENGTH)
    IF (STATUS(1) .LT. 1) GOTO ERROR
    IF (STATUS(1) .NE .2) GOTO NOCSI
                          ; At this point we know we have a CSI sequence
```

8.4.4 New File

This routine requests the name of a new file to be created by displaying the P/OS New File Specification form. The user is allowed to enter the name portion of a filespec. The file type must be in the form .XXX and may not be omitted.

Call

```
XCall DNEWF (status,filename,maxlen,namelen,filetype,
             maxlen,typelen,text,maxlen
             [,msgl,maxlen])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

filename An alpha variable of length 50 (max) which is returned the file name entered by the user.

maxlen A decimal expression indicating the length of the preceding alpha variable or literal.

namelen A decimal variable into which the length of the filename is returned.

filetype An alpha variable of length 4 (max). A default file type must be specified.

typelen A decimal variable in which the length of the file type is returned.

text An alpha expression of up to 72 characters to be displayed at the top of the form.

msg1 An alpha expression of up to 80 characters to be displayed on line 23 of the form.

Example

```
RECORD
    FILNAM      ,A50
    NAMLEN      ,D2
    FILTYP      ,A4,'.DDF'
    TYPLEN      ,D1
    TEXT        ,A35,'Test to check the new file service '
    MESAG       ,A35,'Message at bottom of New File menu.'
    STATUS      ,2D3
PROC
    XCALL DNEWF (STATUS,FILNAM,50,NAMLEN,FILTYP,4,TYPLEN,
&                TEXT,35,MESAG,35)
    IF   (Status(1) .LT. 1) GOTO ERROR
```

8.4.5 Old File

This routine requests the name of one or more existing files by displaying the File Selection Menu. An additional options menu allows the user to select a different directory from which to select a file. The calling task may specify a selection criterion for file names from the default directory, the maximum number of choices the user may make, and several lines of text to be displayed on the menu.

Call

```
XCall DOLDF (status, numchoice, filebuf, sizarr,
             wldcrd, maxlen, text, maxlen,
             [msg1, maxlen, msg2, maxlen])
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

numchoice — A decimal variable indicating the number of choices that the user may make. This variable is updated upon return to indicate how many the user actually selected.

filebuf — An alpha variable or array with numchoice entries. Each entry in the array must be 50 characters in length.

sizarr — A decimal array variable with numchoice entries into which the system will place the actual size of the file names in filebuf.

wldcrd — An alpha expression that specifies the file selection criteria.

maxlen — A decimal expression indicating the length of the preceding alpha variable or literal.

text — An alpha expression containing text to be displayed at the top of the menu.

msg1/2 — Alpha expressions containing text to be displayed at the bottom of the menu.

Example

```
RECORD
    NAMBUF    ,2A50
    SIZARR    ,2D2
    NUMCHS    ,D1,2
    WLDSPC    ,A5,'*.DDF'
    TEXT      ,A35,'Test to check the Old File Service '
    MESAG1    ,A35,'Msg #1 at bottom of New File menu. '
    MESAG2    ,A35,'Msg #2 at bottom of New File menu. '
    STATUS    ,2D3
PROC
    XCALL DOLDF (STATUS,NUMCHS,NAMBUF,SIZARR,WLDSPC,5,
&                TEXT,35,MESAG1,35,MESAG2,35)
    IF   (STATUS(1) .LT. 1) GOTO ERROR
```

8.4.6 Wait for Resume Key

This routine echoes a bell character for all keystrokes except the RESUME key. When the RESUME key is pressed, control returns to the application. The application should display a message such as "Press RESUME to continue." on the screen.

Call

```
XCall DWTRE
```

Note that there are no parameters to this call.

Example

```
RECORD
    TEXT        ,A31,'Please press RESUME to continue'
PROC
    OPEN (1,O,'TT:')
    DISPLAY (1,27,'[24;1H',27,'[J',Text)     ; Clear bottom line
    XCALL DWTRE
```

8.4.7 Send Message to Message/Status Display

This routine sends a message to the P/OS Message/Status Display. This routine enters the specified message into a queue. The end user sees this message when Display Message/Status is selected from the Main Menu.

Call

```
XCall DMSGB (status, message, msglen)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation. See Section 5.9 of the **Tool Kit User's Guide** for status values.

message — An alpha expression containing the message to be sent to the Message/Status Display.

msglen — A decimal expression containing the length of the message parameter.

Example

```
RECORD
    MESAG       ,A80
    STATUS      ,2D3
PROC
    ONERROR     ERR
    OPEN        (1,I,'DZ1:[MYDATA]Important.Fil')
    .
    .
    .
Err,
    MESAG = 'Please insert Data diskette in top drive'
    XCALL DMSGB (STATUS,MESAG,80)
    STOP
```

8.5 CALLABLE P/OS USER SYSTEM SERVICES

Callable P/OS system services include the Communications Facility, PROSE, Print Services, and PRO/SORT.

PROSE and PRO/SORT are callable from Professional Tool Kit DIBOL using information in this section. Print services are available via the PRINT subroutine documented in Appendix C.

The following sections illustrate how to call each of the three facilities from your DIBOL application. Detailed information on these services is documented in Chapter 6 of the **Professional Tool Kit User's Guide**.

8.5.1 PROSE Text Editor

PROSE is the text editor for the Professional. PROSE offers facilities for entering and editing text to create documents, source programs, and memos or similar text files.

By calling the PROSE callable editor task (CET), an application can offer the text editor for use within the context of the application. For example, an electronic mail application might use the editor to provide editing services for message creation or modification. All the editor functions offered to the end user are available in the callable form of the editor.

CALLABLE EDITOR TASK

Call

```
XCall DEDIT (status,infile,maxlen,outfile,maxlen,wkfile,
            maxlen,crenewfil,format,maxline,initleft,
            initright,initwrap,lun)
```

status — A decimal array with two elements used to return a code indicating the results of the requested operation.

See Chapter 6 of the **Tool Kit User's Guide** for possible returned status values.

infile — An alpha expression specifying the input file name.

maxlen — A decimal expression specifying the length of the preceding alpha variable or alpha literal.

outfile — An alpha expression specifying the output file name.

wkfile — An alpha expression containing the name of a temporary file for CET to use.

crenewfil A decimal expression whose value indicates whether to create a new file:

0 = no non-zero = create a new file

format A decimal expression whose value determines whether escape sequences are retained in the output file.

0 = do not retain non-zero = retain

maxline A decimal expression specifying the maximum number of characters on a line that may be saved by a user.

initleft A decimal expression indicating the initial left margin to use when creating a new file.

initright A decimal expression indicating the initial right margin to use when creating a file.

initwrap A decimal expression representing the default setting for word wrap. To disable word wrapping specify a value of 0.

lun A decimal expression containing a DIBOL channel number that is not being used.

Example

```
                                    ;
                                    ; This example will create a new file.
                                    ;
RECORD                              ;
    STATUS     ,2D3                 ; Status variable
    INFIL      ,A12                 ; No Input File
    OUTFIL     ,A12,'NewData.DDF '  ; Output File
    WRKFIL     ,A12,'WorkFile.DDF'  ; Work File
    CREATE     ,D1,1                ; Create new file
    FORMAT     ,D1,0                ; No Esc Sequences
    MAXLIN     ,D2,80               ; Max chars/line
    INILFT     ,D1,1                ; Left Margin
    INIRGT     ,D2,80               ; Right margin
    INIWRP     ,D1,0                ; No word wrapping
    LUN        ,D2,15               ; Free channel
PROC
    XCALL DEDIT (  STATUS, INFIL, 12, OUTFIL, 12, WRKFIL,
&                  12, CREATE, FORMAT, MAXLIN, INILFT,
&                  INIRGT, INIWRP, LUN )
```

8.5.2 PRINT Services

Printing from a DIBOL application is accomplished using the DIBOL PRINT external subroutine. See Appendix C.

8.5.3 PRO/SORT

PRO/SORT is a general-purpose sorting utility that runs on P/OS.

Call

```
XCALL DSORT (filespec,length,status)
```

filespec — An alpha expression containing a PRO/SORT command file specification.

length — A decimal expression containing the number of characters in the filespec.

status — A two-element decimal array that receives status information from PRO/SORT. PRO/SORT error messages may be found in Table 6-3 in **The Tool Kit User's Guide**.

8.6 CALLABLE SYSTEM ROUTINES

This section describes how a DIBOL program may call system routines that are located in a resident library called POSSUM. Details of these routines are in Chapter 8 of the P/OS System Reference Manual. Also, refer to that chapter for possible STATUS values passed back from these routines.

DIBOL programs may call the following POSSUM routines:

- PRODIR -- creates or deletes a directory.
- PROFBI -- formats, initializes, and checks for bad blocks on disks (see restrictions below).
- PROLOG -- translates, creates, and deletes a logical name.
- PROVOL -- mounts, dismounts, bootstraps, and/or writes the bootblock on a volume (see restrictions below).

DIBOL programs may not call or use:

- PROATR -- gets or sets file attributes.

Restrictions on some calls:

- Attribute lists may not be used in calls to PROFBI or in calls to PROVOL.

8.6.1 PRODIR

The PRODIR routine provides two forms of directory manipulation. You can use PRODIR to:

- Create a directory on a device.
- Delete a directory on a device.

DIBOL Call:

```
XCALL DDIR ( Status, Request, File_Name, File_Size )
```

where:

Status is a decimal array with 8 entries (for example: 8D10). See Chapter 8 of the P/OS System Reference Manual for possible status values.

Request is a decimal expression indicating whether to create or to delete a directory.

File_Name is an alpha expression containing a device and directory specification.

File_Size is a decimal expression containing the length of the File_Name argument.

Example:

This example illustrates access to PRODIR from a DIBOL program:

```
RECORD
    ANSWER    ,A1
    STATUS    ,8D10
    DIRECT    ,A50
    REQ       ,D1
    I         ,D1
    ACOUNT    ,A1
    ASTATS    ,A10
PROC
              OPEN (1,O,'TT:')
TOP,          DISPLAY (1,13,10,'CREATE OR DELETE? (C/D) ')
              READS (1,ANSWER)
              IF (ANSWER.EQ.'C') THEN REQ=1 ELSE REQ=2
              DISPLAY (1,13,10,'NAME OF DIRECTORY? ')
              READS (1,DIRECT)
              CALL DDIR ( STATUS, REQ, DIRECT, 50 )
              FOR I FROM 1 THRU 8
                   BEGIN
                   ACOUNT = I
                   ASTATS = STATUS(I)
                   DISPLAY (1,13,10,'STATUS(',ALPHA,') =',ASTATS)
                   END
              GOTO TOP
```

8.6.2 PROFBI

The PROFBI routine provides the mechanism for preparing media on the system. The PROFBI routine allows you to:

- Format a volume.
- Check a volume for bad blocks.
- Initialize a volume.

DIBOL Call:

```
XCALL DFBI ( Status, Request, Dev_Spec, Dev_Size )
```

NOTE

Attribute lists are not supported from DIBOL.

where:

Status	is a decimal array with 8 entries (i.e. 8D10).
Request	is a decimal expression indicating which operation to perform.
Dev_Spec	is an alpha expression containing a device specification.
Dev_Size	is a decimal expression containing the length of the Dev_Spec argument.

Example:

Initialize a diskette in the top drive:

```
RECORD
    STATUS    ,10D8
PROC
    XCALL DFBI ( STATUS, 4, 'DZ1:MYLABEL', 11 )
```

8.6.3 PROLOG

The PROLOG routine provides five forms of logical name manipulation. You can use PROLOG as follows:

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

- Set the default directory and/or device.
- Show the default directory and device.

DIBOL Call:

```
XCall DLOG ( Status, Request, Log_Name, Log_Size
             [,Equiv_Name, Equiv_Size] )
```

where:

Status	is a decimal array with 8 entries (for example, 8D10). See Chapter 8 of the **P/OS System Reference Manual** for possible status values.
Request	is a decimal expression indicating which operation to perform.
Log_Name	is an alpha argument containing or receiving the data. (See the **P/OS System Reference Manual.**)
Log_Size	is a decimal expression containing the length of the Log_Name argument.
Equiv_Name	is an alpha expression containing the device spec to assign to the logical name.
Equiv_Size	is a decimal variable containing the length of the Equiv_Name argument.

Example:

An example to get the system directory that contains my application and set my default there (see next page).

```
RECORD
     MYAPPL     ,A80
     SIZE       ,D2,80
PROC
     XCALL DLOG ( STATUS, 4, 'APPL$DIR', 8, MYAPPL, SIZE )
     XCALL DLOG ( STATUS, 1, MYAPPL, SIZE )
     ...
```

8.6.4 PROVOL

The PROVOL routine provides a two-fold service. You can use the PROVOL routine to mount or dismount volumes. You can also use PROVOL to write a bootblock on a volume and/or bootstrap a volume.

DIBOL XCall:

```
XCALL DVOL ( Status, Request, Dev_Spec, Dev_Size )
```

NOTE

The Attribute List argument is not supported.

where:

Status is a decimal array with 8 entries (for example, 8D10). See Chapter 8 of the **P/OS System Reference Manual** for possible status values.

Request is a decimal expression indicating which operation to perform.

Dev_Spec is an alpha expression containing the device specification of the volume.

Dev_Size is a decimal expression containing the length of the Dev_Name argument.

NOTE

This service may be useful after initializing a diskette. The diskette can be dismounted and remounted without user intervention.

CHAPTER 9

DIBOL INTERFACE TO THE CORE GRAPHICS LIBRARY

This chapter describes how CORE Graphics Library subroutines are called from a DIBOL program. The CORE Graphics Library and the use of the subroutines in that library are explained in the CORE Graphics Library Manual. This chapter is to be used in conjunction with that manual. It assumes knowledge contained in that manual.

9.1 CHAPTER ORGANIZATION

The subsections in this chapter correlate to those in the CORE Graphics Library Manual. The page numbers do not correlate.

NOTE

When the DIBOL calls are individually explained in this chapter, they are preceded by the corresponding BASIC-PLUS-2 call as a means of identifying the call as explained in the CORE Graphics Library Manual.

9.2 THE DIBOL INTERFACE

This section describes how to use the CORE Graphics Library with Professional Host Tool Kit DIBOL and PRO/Tool Kit DIBOL.

9.2.1 Calling CORE Graphics Library Subprograms

Each CGL instruction is a subroutine callable from DIBOL. To transfer control to a CGL subroutine from a DIBOL program or subroutine, use the XCALL statement, which has the format:

```
XCALL DCGL (inst_name [,param,...])
```

inst_name — is a decimal expression specifying the desired CGL instruction. DIBOL provides a file name "CGL.DBL" (listed in Section 9.9) that declares a DIBOL RECORD with fields corresponding to the names of the CGL instructions.

param — is one of up to eight parameters described in the individual instruction sections of this chapter.

NOTE

Any program making a CORE Graphics Library call must use the .INCLUDE directive which appears in the following example.

Example:

```
;
; This program uses the CORE Graphics Library
;
.INCLUDE 'LB:[1,5]CGL.DBL'                ; CGL mnemonics
PROC                                      ;
    XCALL DCGL ( GIC )                    ; Initialize Core
    XCALL DCGL ( GNF )                    ; New Frame
    XCALL DCGL ( GSW, 0, 100, 0, 100 )    ; Set Window
    XCALL DCGL ( GMA2, 0, 0 )             ; Move Absolute 2
    XCALL DCGL ( GRA2, 100, 100 )         ; Rectangle Absolute 2
    XCALL DCGL ( GTC )                    ; Terminate Core
    STOP
```

9.2.2 Building Your DIBOL Program

At this point, you must decide which of the two supplied versions of CGL you intend to use. They are:

- CGLEIS (for the Extended Instruction Set)
- CGLFPU (for the Floating Point Unit)

Some CGL instructions require quite a lot of floating point arithmetic; thus, task images that use CGLFPU will perform better. If possible, supply two task images so that your program is not FPU-dependent.

Compile your program with the DIBOL compiler using the build switch (/B). DIBOL creates a .CMD file and an .ODL file for your program. The .CMD file (for a program named TEST1) looks something like:

```
TEST1/CP=TEST1/MP                          ;
UNITS=24                                   ;
ASG=TI:16                                  ; Foreground terminal
ASG=TI:17                                  ; DDT terminal
ASG=SY:18                                  ; Channel for RENAM/DELET
ASG=TI:19                                  ; POSRES terminal
ASG=SY:23                                  ; Directory Searches
ASG=LB:24                                  ; Message board
                                           ;
CLSTR=DBLRES,POSRES,POSSUM,RMSRES:RO       ;
                                           ;
TASK=TEST1                                 ;
                                           ;
extsct    = mn$buf:4540                    ; static single choice menu
extsct    = dm$buf:4540                    ; dynamic single choice menu
extsct    = mm$buf:1000                    ; multi-choice menu
extsct    = hl$buf:3410                    ; help text/menu
extsct    = fl$buf:4310                    ; file selection/specification
                                           ;
gbldef    = tt$lun:23                      ; (19.) terminal I/O
gbldef    = hl$lun:24                      ; (20.) help frame file
gbldef    = ms$lun:25                      ; (21.) message frame file
gbldef    = mn$lun:26                      ; (22.) menu frame file
gbldef    = wc$lun:27                      ; (23.) directory searches
gbldef    = mb$lin:30                      ; (24.) message board lun
                                           ;
gbldef    = tt$efn:1                       ; terminal I/O event flag
//
```

Make the following edits to the .CMD file:

1. Increment the number of UNITS by 1. It should look like this:

   ```
   UNITS=25
   ```

2. Add the following ASG line prior to the CLSTR line:

   ```
   ASG=TI:25 ; Graphics LUN
   ```

3. Find the line that begins with CLSTR and insert either "CGLEIS" or "CGLFPU". It should look like:

   ```
   CLSTR=DBLRES,CGLEIS,POSRES,POSSUM,RMSRES:RO
       or
   CLSTR=DBLRES,CGLFPU,POSRES,POSSUM,RMSRES:RO
   ```

4. After all of the other GBLDEF lines and add the following line:

   ```
   GBLDEF= G$Lun:31                                ; (25.) Graphics Lun
   ```

5. The .CMD file (with FPU graphics library) should now look like this:

```
TEST1/CP=TEST1/MP                                   ;
UNITS=25                                            ;
ASG=TI:16                                           ; Foreground terminal
ASG=TI:17                                           ; DDT terminal
ASG=SY:18                                           ; Channel for RENAM/DELET
ASG=TI:19                                           ; POSRES terminal
ASG=SY:23                                           ; Directory Searches
ASG=LB:24                                           ; Message board
ASG=TI:25                                           ; Graphics LUN
                                                    ;
CLSTR=DBLRES,CGLFPU,POSRES,POSSUM,RMSRES:RO
                                                    ;
TASK=TEST1                                          ;
                                                    ;
extsct    = mn$buf:4540                             ; static single choice menu
extsct    = dm$buf:4540                             ; dynamic single choice menu
extsct    = mm$buf:1000                             ; multi-choice menu
extsct    = hl$buf:3410                             ; help text/menu
extsct    = fl$buf:4310                             ; file selection/specification
                                                    ;
gbldef    = tt$lun:23                               ; (19.) terminal I/O
gbldef    = hl$lun:24                               ; (20.) help frame file
gbldef    = ms$lun:25                               ; (21.) message frame file
gbldef    = mn$lun:26                               ; (22.) menu frame file
gbldef    = wc$lun:27                               ; (23.) directory searches
gbldef    = tt$efn:1                                ; terminal I/O event flag
gbldef    = mb$lun:30                                     ; (24.) message board lun
GBLDEF    = G$LUN:31                                           ; (25.) Graphics Lun
//
```

9.2.3 Running CORE GRAPHICS Library Programs

Application programs that use CGL must specify the appropriate cluster library in the installation command (.INS) file:

```
INSTALL [ZZSYS]CGLEIS.TSK/LIBRARY
```

or

```
INSTALL [ZZSYS]CGLFPU.TSK/LIBRARY
```

9.3 CONTROL INSTRUCTIONS

This section descibes the instructions that control the overall operation of the CORE Graphics Library.

9.3.1 INITIALIZE_CORE -- Prepare Graphics System for Use

BASIC-PLUS-2 Call

```
CALL CGL BY REF (INITIALIZE_CORE)
```

DIBOL CALL

XCALL DCGL (GIC)

Parameters

GIC defined in CGL.DBL with a value of 90.

9.3.2 TERMINATE_CORE -- Graphics System Usage Finished

BASIC-PLUS-2 CALL

CALL CGL BY REF (TERMINATE_CORE)

DIBOL CALL

XCALL DCGL (GTC)

Parameters

GTC defined in CGL.DBL with a value of 91.

9.3.3 REPORT_MOST_RECENT_ERROR -- Identify Execution Error

BASIC-PLUS-2 CALL

CALL CGL BY REF (REPORT_MOST_RECENT_ERROR ,inst_name,code)

DIBOL CALL

XCALL DCGL (GRMRE,inst_name,code)

Parameters

GRMRE defined in CGL.DBL with a value of 93.

inst_name decimal field to which is returned the instruction number (defined in CGL.DBL) of the call that received the error.

code decimal field to receive the value.

9.3.4 NEW_FRAME -- Refresh Screen

BASIC-PLUS-2 CALL

CALL CGL BY REF (NEW_FRAME)

DIBOL CALL

```
XCALL DCGL (GNF)
```

Parameters

GNF defined in CGL.DBL with a value of 92.

9.3.5 ERASE VIEWPORT -- Erase Images in Viewport

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (ERASE_VIEWPORT)
```

DIBOL CALL

```
XCALL DCGL (GEV)
```

Parameters

GEV defined in CGL.DBL with a value of 88.

9.3.6 PRINT_SCREEN -- Send Screen Image to Output Device

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (PRINT_SCREEN,lower_x,upper_x,
                 lower_y,upper_y,x_offset,y_offset)
```

DIBOL CALL

```
XCALL DCGL (GPS,lower_x,upper_x,lower_y,upper_y,x_offset,y_offset)
```

Parameters

All parameters in this call are decimal fields or literals.

GPS	defined in CGL.DBL with a value of 94.
lower_x	specifies the lower limit of x
upper_x	specifies the upper limit of x
lower_y	specifies the lower limit of y
upper_y	specifies the upper limit of y
x_offset	specifies the horizontal margin.
y_offset	specifies the vertical margin.

9.3.7 CGL_WAIT -- Suspend Execution

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (CGL_WAIT,seconds)
```

DIBOL CALL

```
XCALL DCGL (GCW,seconds)
```

Parameters

GCW	defined in CGL.DBL with a value of 95.
seconds	decimal expression specifying the number of seconds.

9.4 VIEWING TRANSFORMATION INSTRUCTIONS

9.4.1 SET_WINDOW - Specify Visible Part of World Coordinate Space

BASIC-PLUS-2 Call

```
CALL CGL BY REF (SET_WINDOW,lower_x,upper_x,lower_y,upper_y)

CALL CGL BY REF (INQUIRE_WINDOW,lower_x,upper_x,lower_y,upper_y)
```

DIBOL CALL

```
XCALL DCGL (GSW,lower_x,upper_x,lower_y,upper_y)

XCALL DCGL (GIW,lower_x,upper_x,lower_y,upper_y)
```

Parameters

The set parameters are decimal expressions while the inquire parameters are decimal variables.

GSW,GIW	defined in CGL.DBL with values of 80,81.
lower_x	specifies the X lower limit of the window
upper_x	specifies the X upper limit of the window
lower_y	specifies the Y lower limit of the window
upper_y	specifies the Y upper limit of the window

9.4.2 SET_ORIGIN -- Specify Origin of Window

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_ORIGIN,origin)

CALL CGL BY REF (INQUIRE_ORIGIN,origin)
```

DIBOL CALL

XCALL DCGL (GSO,origin)

Parameters

GSO,GIO defined in CGL.DBL with values of 86,87.

origin a decimal expression/variable to set or receive the origin.

9.4.3 SET_WINDOW_CLIPPING -- Enable or Disable Window Clipping

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_WINDOW_CLIPPING,on_off)

CALL CGL BY REF (INQUIRE_WINDOW_CLIPPING, on_off)

DIBOL CALL

XCALL DCGL (GSWC,on_off)

XCALL DCGL (GIWC,on_off)

Parameters

GSWC,GIWC defined in CGL.DBL with values of 84,85.

on_off a decimal expression/variable to set or receive the window clipping flag. Zero is OFF, any other value is ON.

9.4.4 SET_VIEWPORT_2 -- Specify Usable Area of Screen

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_VIEWPORT_2,lower_x,upper_x,lower_y,upper_y)

CALL CGL BY REF (INQUIRE_VIEWPORT_2,lower_x,lower_y,lower_y,upper_y)

NOTE

Set/Inquire viewport requires special attention, because the calls require numbers between zero and one that are not supported in DIBOL. These two calls are implemented with with a 'division factor'. The interface will convert all of the arguments to divide or multiply the x and y values by the factor before calling CGL.

```
XCALL DCGL (GSV2,lower_x,upper_x,lower_y,upper_y,Factor)

XCALL DCGL (GIV2,lower_x,upper_x,lower_y,upper_y[,Factor])
```

Parameters

All parameters for the Set Viewport are decimal expressions, but in the Inquire call all parameters are decimal variables except for the first (GIV2).

```
GSV2,GIV2   defined in CGL.DBL with values of 82,83.
lower_x     specifies the lower limit of X
upper_x     specifies the upper limit of X
lower_y     specifies the lower limit of Y
upper_y     specifies the upper limit of Y
```

Example

```
        ; Set the viewport 4 times, each time defining a
        ; different quartile.
        ;
        ;          X
        ;  0                1
        ;  +-------+-------+ 0
        ;  |   1   |   4   |
        ;  +-------+-------+   Y
        ;  |   2   |   3   |
        ;  +-------+-------+ 1

PROC
        XCALL DCGL (GIC)
        XCALL DCGL (GSV2, 0, 5, 0, 5, 10)              ; Quartile # 1
        XCALL DCGL (GSV2, 0, 5, 5, 10, 10)             ; Quartile # 2
        XCALL DCGL (GSV2, 5, 10, 5, 10, 10)            ; Quartile # 3
        XCALL DCGL (GSV2, 5, 10, 0, 5, 10)             ; Quartile # 4
```

9.4.5 SCROLL -- Move Screen Contents

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SCROLL,delta_x,delta_y)
```

DIBOL CALL

```
XCALL DCGL (GS,delta_x,delta_y)
```

Parameters

All parameters in this call are decimal expressions.

GS	defined in CGL.DBL with a value of 89.
delta_x	specifies the X (horizontal) movement.
delta_y	specifies the Y (horizontal) movement.

9.5 GLOBAL ATTRIBUTE INSTRUCTIONS

9.5.1 SET_WRITING_INDEX -- Select Color Map Index for Images

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_WRITING_INDEX,index)
CALL CGL BY REF (INQUIRE_WRITING_INDEX,index)
```

DIBOL CALL

```
XCALL DCGL (GSWI,index)
XCALL DCGL (GIWI,index)
```

Parameters

GSWI,GIWI	defined in CGL.DBL with values of 60,61.
index	a decimal expression/variable used to set or inquire the writing index.

9.5.2 SET_BACKGROUND_INDEX -- Set Background Color Map Index

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_BACKGROUND_INDEX,index)
CALL CGL BY REF (INQUIRE_BACKGROUND_INDEX,index)
```

DIBOL CALL

XCALL DCGL (GSBI,index)

XCALL DCGL (GIWI,index)

Parameters

GSBI,GIBI defined in CGL.DBL with values of 62,63.

index a decimal expression/variable used to set or inquire the background index.

9.5.3 SET_COLOR_MAP_ENTRY -- Set Color Map Entry RGB Values

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_COLOR_MAP_ENTRY,entry,color)

CALL CGL BY REF (INQUIRE_COLOR_MAP_ENTRY,entry,color)

DIBOL CALL

XCALL DCGL (GSCME,entry,color)

XCALL DCGL (GICME,entry,color)

Parameters

GSCME,GICME defined in CGL.DBL with values of 66 and 67.

entry a decimal expression specifying which color map entry (0-7) to set or inquire.

color a decimal expression/variable that contains or will receive the index.

Example

```
RECORD
    WHITE,    3D1,7,7,6
    BLACK,    3D1,0,0,0
PROC
    XCALL DCGL (GIC)              ; Init CGL
    XCALL DCGL (GSCME,4,WHITE)    ; Color 5 is white
    XCALL DCGL (GSCME,0,BLACK)    ; Color 1 is black
```

9.5.4 SET_COLOR_MAP -- Set All Color Map RGB Values

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_COLOR_MAP,color_map)

CALL CGL BY REF (INQUIRE_COLOR_MAP,color_map)
```

DIBOL CALL

```
XCALL DCGL (GSCM,color_map)

XCALL DCGL (GICM,color_map)
```

Parameters

GSCM,GICM defined in CGL.DBL with values of 64,65.

color_map a 24 element decimal array containing colors (range 0 to 7) that specify all eight color map entries.

Example:

```
RECORD
OLDMAP,  24D1
NEWMAP,  24D1,7,0,0 ,0,7,0 ,0,0,6 ,0,0,0
&             ,7,7,6 ,7,7,0 ,7,0,6 ,0,7,6
PROC
    XCALL DCGL (GIC)          ; Must always be done
    XCALL DCGL (GICM,OLDMAP)  ; Fetch the old map
    XCALL DCGL (GSCM,NEWMAP)  ; Replace with our map
    ...                       ; Intervening code
    XCALL DCGL (GSCM,OLDMAP)  ; Restore old color map
    XCALL DCGL (GTC)          ; And terminate core
    STOP                      ;
```

9.5.5 SET_WRITING_PLANES -- Select Combination of Planes

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_WRITING_PLANES,n)

CALL CGL BY REF (INQUIRE_WRITING_PLANES,n)
```

DIBOL CALL

```
XCALL DCGL (GSWP,n)

XCALL DCGL (GIWP,n)
```

Parameters

GSWP,GIWP defined in CGL.DBL with values of 68,69.

n a decimal expression/variable that sets or receives the plane combination value.

9.5.6 SET_WRITING_MODE -- Set Writing Characteristics

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_WRITING_MODE,mode)

CALL CGL BY REF (INQUIRE_WRITING_MODE,mode)
```

DIBOL CALL

```
XCALL DCGL (GSWM,mode)

XCALL DCGL (GIWM,mode)
```

Parameters

GSWM,GIWM defined in CGL.DBL with values of 70,71.

mode a decimal expression/variable that sets or receives the writing mode.

9.5.7 SET_GLOBAL_ATTRIBUTES -- Set Global Attribute List

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_GLOBAL_ATTRIBUTES,int_list,real_list)

CALL CGL BY REF (INQUIRE_GLOBAL_ATTRIBUTES,int_list,real_list)
```

DIBOL CALL

```
XCALL DCGL (GSGA,int_list,real_list)

XCALL DCGL (GIGA,int_list,real_list)
```

Parameters

GSGA,GIGA defined in CGL.DBL with values of 72,73.

int_list a decimal variable array with 19 entries as shown in the CGL manual.

real_list an 8 entry decimal variable array with entries as shown in the CGL manual.

9.6 CURRENT POSITION AND MARKER INSTRUCTIONS

9.6.1 Current Position Instructions

9.6.1.1 MOVE_ABSOLUTE_2 -- Move to Absolute Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (MOVE_ABSOLUTE_2,x,y)
```

DIBOL CALL

```
XCALL DCGL (GMA2,x,y)
```

Parameters

GMA2 defined in CGL.DBL with a value of 1.

x,y decimal expressions specifying the x and y values.

9.6.1.2 MOVE_REL_2 - Move to Relative Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (MOVE_REL_2,delta_x,delta_y)
```

DIBOL CALL

```
XCALL DCGL (GMR2,delta_x,delta_y)
```

Parameters

GMR2 defined in CGL.DBL with a value of 2.

delta_x a decimal expression specifying the change in the X (horizontal) position.

delta_y a decimal expression specifying the change in the Y (vertical) position.

9.6.1.3 INQUIRE_CURRENT_POSITION_2 -- Get Current Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (INQUIRE_CURRENT_POSITION_2,x,y)
```

DIBOL CALL

```
XCALL DCGL (GICP2,x,y)
```

Parameters

GICP2	defined in CGL.DBL with a value of 3.
x,y	decimal variables to receive the coordinates.

9.6.2 Marker Primitive Instructions

9.6.2.1 MARKER_ABS_2 -- Draw Marker at Absolute Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (MARKER_ABS_2,x,y)
```

DIBOL CALL

```
XCALL DCGL (GMKA2,x,y)
```

Parameters

GMKA2	defined in CGL.DBL with a value of 33.
x,y	decimal expressions containing the x and y coordinates of where to put the marker.

9.6.2.2 MARKER_REL_2 -- Draw Marker Relative to Current Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (MARKER_REL_2,delta_x,delta_y)
```

DIBOL CALL

```
XCALL DCGL (GMKR2,delta_x,delta_y)
```

Parameters

GMKR2	defined in CGL.DBL with a value of 34.

delta_x — a decimal expression specifying the X offset at which to draw a marker.

delta_y — a decimal expression specifying the Y offset at which to draw a marker.

9.6.2.3 POLYMARKER_ABS_2 -- Draw Markers at Absolute Positions

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (POLYMARKER_ABS_2,x_array,y_array,n)
```

DIBOL CALL

```
XCALL DCGL (GPMA2, x_array, y_array, n)
```

Parameters

GPMA2 — defined in CGL.DBL with a value of 35.

x_array — a decimal variable array with 'n' entries specifying the X coordinates of where to draw the markers.

y_array — a decimal variable array with 'n' entries specifying the Y coordinates of where to draw the markers.

n — a decimal expression specifying the number of valid entries in the x_array and y_array parameters.

NOTE

When an array is specified in this manner it is imperative that the value of 'n' be greater than zero and less than or equal to the actual size of the array.

Example

```
RECORD
    X,   6D2,10 ,40 ,90 ,30 ,20 ,50
    Y,   6D2,30 ,20 ,10 ,30 ,60 ,90
PROC
    XCALL DCGL (GIC)          ; Initialize CGL
    XCALL DCGL (GPMA2,X,Y,6) ; Draw the Markers
    ...
```

9.6.2.4 POLYMARKER_REL_2 -- Draw Markers at Relative Positions

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (POLYMARKER_REL_2,dx_array,dy_array,n)
```

DIBOL CALL

XCALL DCGL (GPMR2,dx_array,dy_array,n)

Parameters

GPMR2	defined in CGL.DBL with a value of 36.
dx_array	a decimal variable array with 'n' entries specifying the X offsets at which to draw the markers.
dy_array	a decimal variable array with 'n' entries specifying the Y offsets at which to draw the markers.
n	a decimal expression containing the number of entries in the two arrays.

9.6.3 Marker Attribute Instuctions

9.6.3.1 SET_MARKER_SYMBOL -- Select New Marker Symbol

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_MARKER_SYMBOL,symbol,code)

CALL CGL BY REF (INQUIRE_MARKER_SYMBOL,symbol,code)

DIBOL CALL

XCALL DCGL (GSMKS,symbol,code)

XCALL DCGL (GIMKS,symbol,code)

Parameters

GSMKS,GIMKS	defined in CGL.DBL with values of 37 and 38.
symbol	a decimal expression/variable used to set or receive the marker symbol.
code	a decimal expression/variable used to set or receive the decimal code of a user defined marker symbol.

9.7 LINE INSTRUCTIONS

9.7.1 Straight Line Primitive Instructions

9.7.1.1 LINE_ABS_2 -- Draw Line to Absolute Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (LINE_ABS_2,x,y)
```

DIBOL CALL

```
XCALL DCGL (GLA2,z,y)
```

Parameters

GLA2	defined in CGL.DBL with a value of 4.
x,y	decimal expressions specifying the X and Y coordinates to which a line is to be drawn.

9.7.1.2 LINE_REL_2 -- Draw Line to Relative Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (LINE_REL_2,delta_x,delta_y)
```

DIBOL CALL

```
XCALL DCGL (GLR2,delta_x,delta_y)
```

Parameters

GLR2	defined in CGL.DBL with a value of 5.
delta_x,y	decimal expressions specifying the X and Y offsets to which a line is to be drawn.

9.7.1.3 POLYLINE_ABS_2 -- Draw Lines to Absolute Positions

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (POLYLINE_ABS_2,x_array,y_array,n)
```

DIBOL CALL

```
XCALL DCGL (GPLA2,x_array,y_array,n)
```

Parameters

GPLA2 defined in CGL.DBL with a value of 6.

x_array a decimal variable array with 'n' entries specifying the X entries to which a line is to be drawn.

y_array a decimal variable array with 'n' entries specifying the Y entries to which a line is to be drawn.

n a decimal expression specifying the number of entries in the x_array and y_array parameters.

9.7.1.4 POLYLINE_REL_2 -- Draw Lines to Relative Positions

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (POLYLINE_REL_2,dx_array,dy_array,n)
```

DIBOL CALL

```
XCALL DCGL (GPLR2,dx_array,dy_array,n)
```

Parameters

GPLR2 defined in CGL.DBL with a value of 7.

dx_array a decimal variable array with 'n' entries specifying the X offsets to which a line is to be drawn.

dy_array a decimal variable array with 'n' entries specifying the Y offsets to which a line is to be drawn.

n a decimal expression specifying the number of entries in each of the arrays.

9.7.1.5 POLYGON_ABS_2 -- Draw Polygon by Absolute Positions

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (POLYGON_ABS_2,x_array,y_array,n)
```

DIBOL CALL

```
XCALL DCGL (GPGA2,x_array,y_array,n)
```

Parameters

GPGA2 defined in CGL.DBL with a value of 8.

x_array — a decimal variable array containing a list of the X coordinates describing the polygon.

y_array — a decimal variable array containing a list of the Y coordinates describing the polygon.

n — a decimal expression specifying the number of entries in each of the two arrays.

9.7.1.6 POLYGON_REL_2 -- Draw Polygon by Relative Positions

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (POLYGON_REL_2,dx_array,dy_array,n)
```

DIBOL CALL

```
XCALL DCGL (GPGR2,dx_array,dy_array,n)
```

Parameters

GPGR2 — defined in CGL.DBL with a value of 9.

dx_array — a decimal variable array containing a list of the X coordinate offsets describing the polygon.

dy_array — a decimal variable array containing a list of the Y coordinate offsets describing the polygon.

n — a decimal expression specifying the number of entries in each of the two arrays.

9.7.1.7 RECTANGLE_ABS_2 -- Draw Rectangle by Absolute Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (RECTANGLE_ABS_2,x,y)
```

DIBOL CALL

```
XCALL DCGL (GRA2,x,y)
```

Parameters

GRA2 — defined in CGL.DBL with a value of 10.

x,y — decimal expressions specifying the opposite corner (the current position being the other) of the rectangle.

9.7.1.8 RECTANGLE_REL_2 -- Draw Rectangle by Relative Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (RECTANGLE_REL_2,dx,dy)
```

DIBOL CALL

```
XCALL DCGL (GRR2,dx,dy)
```

Parameters

GRR2	defined in CGL.DBL with a value of 11.
dx,dy	decimal expressions containing the X and Y offsets to the other corner of the rectangle.

9.7.2 Curved Line Primitive Instructions

9.7.2.1 ARC_ABS_2 -- Draw Arc Based on Absolute Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (ARC_ABS_2,x,y,angle)
```

DIBOL CALL

```
XCALL DCGL (GAA2,x,y,angle)
```

Parameters

GAA2	defined in CGL.DBL with a value of 39.
x,y	decimal expressions containing the coordinates of the center of the arc.
angle	a decimal expression containing the number of degrees to draw the arc.

9.7.2.2 ARC_REL_2 -- Draw Arc Based on Relative Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (ARC_REL_2,x,y,angle)
```

DIBOL CALL

```
XCALL DCGL (GAR2,x,y,angle)
```

Parameters

GAR2 defined in CGL.DBL with a value of 40.

x,y decimal expressions containing the offset from the current position of the center of the arc.

angle a decimal expression containing the number of degrees to draw the arc.

9.7.2.3 CURVE_ABS_2 -- Draw Curve by Absolute Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (CURVE_ABS_2,x_array,y_array,n,type)
```

DIBOL CALL

```
XCALL DCGL (GCA2,x_array,y_array,n,type)
```

Parameters

GCA2 defined in CGL.DBL with a value of 41.

x_array a decimal variable array with 'n' entries with the X coordinates of the curve.

y_array a decimal variable array with 'n' entries with the Y coordinates of the curve.

n a decimal expression containing the number of entries in the above arrays.

type a decimal expression that specifies either an open curve (type=0) or a closed curve (type=non-zero).

9.7.2.4 CURVE_REL_2 -- Draw Curve by Relative Position

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (CURVE_REL_2,x_array,y_array,n,type)
```

DIBOL CALL

```
XCALL DCGL (GCR2,x_array,y_array,n,type)
```

Parameters

GCR2 defined in CGL.DBL with a value of 42.

x_array — a decimal variable array with 'n' entries specifying the X coordinate offsets of the curve.

y_array — a decimal variable array with 'n' entries specifying the Y coordinate offsets of the curve.

n — a decimal expression containing the number of entries in the above arrays.

type — a decimal expression that specifies either an open curve (type=0) or a closed curve (type=non-zero).

9.7.3 Line Attribute Instructions

9.7.3.1 SET_LINESTYLE -- Set Line Drawing Style

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_LINESTYLE,style,pattern,mult)

CALL CGL BY REF (INQUIRE_LINESTYLE,style,pattern,mult)
```

DIBOL CALL

```
XCALL DCGL (GSLS,style,pattern,mult)

XCALL DCGL (GILS,style,pattern,mult)
```

Parameters

GSLS,GILS — defined in CGL.DBL with values of 12,13.

style — a decimal expression/variable specifying or receiving one of the nine standard line styles or a user defined style.

pattern — an alpha expression/variable with a length of 16 characters representing a 'bit pattern' for the line style. Spaces and Zeros are OFF, anything else is ON.

mult — a decimal expression/variable specifying how many times to draw each bit in the pattern.

9.7.3.2 SET_LINEWIDTH -- Set Line Drawing Width

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_LINEWIDTH,dx,dy)

CALL CGL BY REF (INQUIRE_LINEWIDTH,dx,dy)
```

DIBOL CALL

XCALL DCGL (GSLW,dx,dy)

XCALL DCGL (GILW,dx,dy)

Parameters

GSLW,GILW defined in CGL.DBL with values of 14,15.

dx,dy decimal expressions/variables to specify or receive the horizontal and vertical width of lines created by primitive line instructions.

9.7.3.3 SET_FILL_MODE -- Enable or Disable Area Fill

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_FILL_MODE,mode)

CALL CGL BY REF (INQUIRE_FILL_MODE,mode)

DIBOL CALL

XCALL DCGL (GSFM,mode)

XCALL DCGL (GIFM,mode)

Parameters

GSFM,GIFM defined in CGL.DBL with values of 74,75.

mode a decimal expression/variable that sets or receives the fill mode indicator.

9.7.3.4 SET_FILL_ENTITY -- Set Fill to Line or Point

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_FILL_ENTITY,x,y)

CALL CGL BY REF (INQUIRE_FILL_ENTITY,x,y)

DIBOL CALL

XCALL DCGL (GSFE,x,y)

XCALL DCGL (GIFE,x,y)

Parameters

GSFE,GIFE defined in CGL.DBL with values of 76,77.

x,y decimal expressions/variables that set or receive the fill entity coordinates.

9.7.3.5 SET_FILL_CHAR -- Specify Character for Fill

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_FILL_CHARACTER,font,char,width_mult,height_mult)

CALL CGL BY REF (INQUIRE_FILL_CHARACTER,font,char,width_mult
                 ,height_mult)
```

DIBOL CALL

```
XCALL DCGL (GSFC,font,char,width_mult,height_mult)

XCALL DCGL (GIFC,font,char,width_mult,height_mult)
```

Parameters

GSFC,GIFC defined in CGL.DBL with a value of 78.

font a decimal expression/variable that sets or receives the font number that the fill character is in.

char a decimal expression/variable that sets or receives the numeric code for the fill character.

width_mult a decimal expression/variable that sets or receives the width multiplier for the fill character.

height_mult a decimal expression/variable that sets or receives the height multiplier for the fill character.

9.8 TEXT INSTRUCTIONS

9.8.1 Text Primitive Instructions

9.8.1.1 TEXT -- Draw Line of Text

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (TEXT,string,length)
```

DIBOL CALL

XCALL DCGL (GT,string,length)

Parameters

GT — defined in CGL.DBL with a value of 16.

string — an alpha expression containing the text to be displayed.

length — a decimal expression specifying the number of characters in the string argument.

9.8.1.2 INQUIRE_TEXT_EXTENT_2 -- Report Position at End of String

BASIC-PLUS-2 CALL

CALL CGL BY REF (INQUIRE_TEXT_EXTENT_2,length,dx,dy)

DIBOL CALL

XCALL DCGL (GITE2,length,dx,dy)

Parameters

GITE2 — defined in CGL.DBL with a value of 17.

length — a decimal expression specifying the number of characters in the string.

dx,dy — decimal variables to receive the X and Y offsets.

9.8.2 Text Attribute Instructions

9.8.2.1 SET_CHARSIZE -- Set Character Size

BASIC-PLUS-2 CALL

CALL CGL BY REF (SET_CHARSIZE,width,height)

CALL CGL BY REF (INQUIRE_CHARSIZE,width,height)

DIBOL CALL

XCALL DCGL (GSCS,width,height)

XCALL DCGL (GICS,width,height)

Parameters

GSCS,GICS defined in CGL.DBL with values of 20,21.

width A decimal expression/variable specifying the character width.

height A decimal expression/variable specifying the character height.

9.8.2.2 SET_CHARSPACE -- Set Character Spacing

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_CHARSPACE,delta_x,delta_y)

CALL CGL BY REF (INQUIRE_CHARSPACE,delta_x,delta_y)
```

DIBOL CALL

```
XCALL DCGL (GSCSP,delta_x,delta_y)

XCALL DCGL (GICSP,delta_x,delta_y)
```

Parameters

GSCSP,GICSP defined in CGL.DBL with values of 24 and 25.

delta_x a decimal expression/variable to set or to receive the X (horizontal) character spacing.

delta_y a decimal expression/variable to set or to receive the Y (vertical) character spacing.

9.8.2.3 SET_CHARPATH -- Set Text Writing Direction

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_CHARPATH,path,mode)

CALL CGL BY REF (INQUIRE_CHARPATH,path,mode)
```

DIBOL CALL

```
XCALL DCGL (GSCP,path,mode)

XCALL DCGL (GICP,path,mode)
```

Parameters

GSCP,GICP defined in CGL.DBL with values of 22,23.

path decimal expression/variable to set or to receive the character path.

mode decimal expression/variable to set or to receive the character path mode.

9.8.2.4 SET_CHARJUST -- Set Text Justification

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_CHARJUST,x_just,y_just)

CALL CGL BY REF (INQUIRE_CHARJUST,x_just,y_just)
```

DIBOL CALL

```
XCALL DCGL (GSCJ,x_just,y_just)

XCALL DCGL (GICJ,x_just,y_just)
```

Parameters

GSCJ,GICJ defined in CGL.DBL with values of 26,27.

x_just decimal expression/variable to set or to receive the X justification value.

y_just decimal expression/variable to set or to receive the Y justification value.

9.8.2.5 SET_CHARITALIC -- Set Character Slant

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_CHARITALIC,angle)

CALL CGL BY REF (INQUIRE_CHARITALIC,angle)
```

DIBOL CALL

```
XCALL DCGL (GSCI,angle)

XCALL DCGL (GICI,angle)
```

Parameters

GSCI,GICI defined in CGL.DBL with a value of 28,29.

angle decimal expression/variable to set or to receive the angle of italics.

9.8.2.6 SET_FONT -- Set Character Font

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_FONT, font)

CALL CGL BY REF (INQUIRE_FONT,font)
```

DIBOL CALL

```
XCALL DCGL (GSF,font)

XCALL DCGL (GIF,font)
```

Parameters

GSF,GIF defined in CGL.DBL with values of 18,19.

font a decimal expression or variable in the case of Inquire to set or to receive the current font that is being used.

9.8.2.7 SET_FONT_SIZE -- Define Size of Character Font

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (SET_FONT_SIZE,extent,x_size,y_size)

CALL CGL BY REF (INQUIRE_FONT_SIZE,extent,x_size,y_size)
```

DIBOL CALL

```
XCALL DCGL (GSFS,extent,x_size,y_size)

XCALL DCGL (GIFS,extent,x_size,y_size)
```

Parameters

GSFS,GIFS defined in CGL.DBL with values of 30,31.

extent a decimal expression/variable to set or receive the highest numbered character in the font.

x_size — a decimal expression/variable to set or receive the width (x size) of the font characters.

y_size — a decimal expression/variable to set or receive the height of the characters.

9.8.2.8 LOAD_CHARACTER -- Load User-defined Character

BASIC-PLUS-2 CALL

```
CALL CGL BY REF (LOAD_CHARACTER,code,matrix)
```

DIBOL CALL

```
XCALL DCGL (GLC,code,matrix)
```

Parameters

GLC — defined in CGL.DBL with a value of 32.

code — a decimal expression specifying which character of the font is being loaded.

matrix — an alpha variable array with 16 entries; each entry is 16 characters in length.

Example

```
RECORD
    MYA  ,16A16,'  A  '
&               ,' A A '
&               ,'A   A'
&               ,'AAAAA'
&               ,'A   A'
&               ,'A   A'
&               ,'A   A'
&               ,'A   A'

PROC
    XCALL DCGL (GIC)          ; Initialize CGL
    XCALL DCGL (GSF,3)        ; Font 3
    XCALL DCGL (GSFS,126,5,8); Set the size
    XCALL DCGL (GLC,65,MyA)   ; Load character
        ...
```

9.9 CONSTANT DECLARATION FILE

This section is a listing of the constant declaration file (CGL.DBL) with the values for DIBOL which correspond to a similar file used by CORE graphics.

```
;
; Control Instructions
;
;
RECORD                      ;
    GIC         ,D2   ,90   ; Initialize cgl
    GTC         ,D2   ,91   ; Terminate cgl
    GRMRE       ,D2   ,93   ; Report most recent error
    GNF         ,D2   ,92   ; New frame
    GEV         ,D2   ,88   ; Erase viewport
    GPS         ,D2   ,94   ; Print screen
    GCW         ,D2   ,95   ; Cgl wait
;
;
; Global Attribute Instructions
;
;
    GSWI        ,D2   ,60   ; Set writing index
    GIWI        ,D2   ,61   ; Inquire writing index
    GSBI        ,D2   ,62   ; Set background index
    GIBI        ,D2   ,63   ; Inquire background index
    GSCME       ,D2   ,66   ; Set color map entry
    GICME       ,D2   ,67   ; Inquire color map entry
    GSCM        ,D2   ,64   ; Set color map
    GICM        ,D2   ,65   ; Inquire color map
    GSWP        ,D2   ,68   ; Set writing planes
    GIWP        ,D2   ,69   ; Inquire writing planes
    GSWM        ,D2   ,70   ; Set writing mode
    GIWM        ,D2   ,71   ; Inquire writing mode
    GSGA        ,D2   ,72   ; Set global attributes
    GIGA        ,D2   ,73   ; Inquire global attributes
;
;
; Viewing Transformation Instructions
;
;
    GSV2        ,D2   ,82   ; Set viewport 2
    GIV2        ,D2   ,83   ; Inquire viewport 2
    GSW         ,D2   ,80   ; Set window
    GIW         ,D2   ,81   ; Inquire window
    GSO         ,D2   ,86   ; Set origin
    GIO         ,D2   ,87   ; Inquire origin
    GSWC        ,D2   ,84   ; Set window clipping
    GIWC        ,D2   ,85   ; Inquire window clipping
    GS          ,D2   ,89   ; Scroll
;
;
; Current Position Instructions
;
;
    GMA2        ,D2   ,01   ; Move Absolute 2
    GMR2        ,D2   ,02   ; Move Relative 2
    GICP2       ,D2   ,03   ; Inquire Current Position 2
                            ;
```

```
;
;
; Marker Instructions
;
;
    GMKA2       ,D2   ,33   ; Marker absolute 2
    GMKR2       ,D2   ,34   ; Marker relative 2
    GPMA2       ,D2   ,35   ; Polymarker absolute 2
    GPMR2       ,D2   ,36   ; Polymarker relative 2
;
;
; Marker Attributes
;
;
    GSMKS       ,D2   ,37   ; Set marker symbol
    GIMKS       ,D2   ,38   ; Inquire marker symbol
;
;
; Straight Line-Drawing Instructions
;
;
    GLA2        ,D2   ,04   ; Line absolute 2
    GLR2        ,D2   ,05   ; Line relative 2
    GPLA2       ,D2   ,06   ; Polyline absolute 2
    GPLR2       ,D2   ,07   ; Polyline relative 2
    GPGA2       ,D2   ,08   ; Polygon absolute 2
    GPGR2       ,D2   ,09   ; Polygon relative 2
    GRA2        ,D2   ,10   ; Rectangle absolute 2
    GRR2        ,D2   ,11   ; Rectangle relative 2
;
;
; Curved Line-Drawing Instructions
;
;
    GAA2        ,D2   ,39   ; Arc absolute 2
    GAR2        ,D2   ,40   ; Arc relative 2
    GCA2        ,D2   ,41   ; Curve absolute 2
    GCR2        ,D2   ,42   ; Curve relative 2
;
;
; Line Attributes
;
;
    GSLS        ,D2   ,12   ; Set line style
    GILS        ,D2   ,13   ; Inquire line style
    GSLW        ,D2   ,14   ; Set line width
    GILW        ,D2   ,15   ; Inquire line width
    GSFM        ,D2   ,74   ; Set fill mode
    GIFM        ,D2   ,75   ; Inquire fill mode
    GSFE        ,D2   ,76   ; Set fill entity
    GIFE        ,D2   ,77   ; Inquire fill entity
    GSFC        ,D2   ,78   ; Set fill character
    GIFC        ,D2   ,79   ; Inquire fill character
```

```
;
;
; Text Primitive Instructions
;
;
    GT          ,D2   ,16   ; Text
    GITE2       ,D2   ,17   ; Inquire text extent 2
;
;
; Text Attribute Instructions
;
;
    GSCS        ,D2   ,20   ; Set character size
    GICS        ,D2   ,21   ; Inquire character size
    GSCSP       ,D2   ,24   ; Set character space
    GICSP       ,D2   ,25   ; Inquire character space
    GSCP        ,D2   ,22   ; Set character path
    GICP        ,D2   ,23   ; Inquire character path
    GSCJ        ,D2   ,26   ; Set character justification
    GICJ        ,D2   ,27   ; Inquire char justification
    GSCI        ,D2   ,28   ; Set character italics
    GICI        ,D2   ,29   ; Inquire character italics
    GSF         ,D2   ,18   ; Set font
    GIF         ,D2   ,19   ; Inquire font
    GSFS        ,D2   ,30   ; Set font size
    GIFS        ,D2   ,31   ; Inquire font size
    GLC         ,D2   ,32   ; Load character
```

9.10 EXAMPLES

This section consists of the DIBOL version of a program that exists in the CORE graphics manual. The program allows you to generate graphics in an interractive mode at a terminal.

GEDIT -- GRAPHICS SKETCHPAD

```
;
; GEDIT - GRAPHICS SKETCHPAD
;
; Instructions:
;
; The status line displays the current function, home position,
; and mode.
;
;
```

Graphics sketchpad (cont.)

```
;                       ***  Editing Keys  ***
;
; +----+----+----+      MOVE: Select move mode
; |MOVE|WRIT|ERAS|      WRIT: Select write mode
; +----+----+----+      ERAS: Select erase mode
; | NH |HOME| CS |      NH:   Set new home position
; +----+----+----+      HOME: Return to home position
;      | ^  |           CS:   Clear screen
; +----+----+----+
; | <- | V  | -> |      The arrow keys move the cursor
; +----+----+----+
;
;
;                       ***  Function Keys  ***
;
;  F17  F18  F19  F20
; +----+----+----+----+      VECT: Begin vector
; |VECT|RECT|CIRC| PS |      RECT: Begin rectangle
; +----+----+----+----+      CIRC: Begin circle
;                            PS:   Print screen
;
;
;
.PAGE                                           ;
.INCLUDE 'LB:[1,5]CGL.DBL'                      ; CGL symbols
RECORD                                          ;
    XINCR       ,D1   ,3                        ; X movement Index
    YINCR       ,D1   ,-2                       ; Y movement Index
    CIRCUM      ,D3   ,360                      ; Degrees in a circle
RECORD STATS                                    ;
    ESC1        ,A1                             ;
                ,A6   ,'[24;1H'                 ;
                ,A7   ,'Home = '                ;
    STSX        ,A4                             ;
                ,A1   ,','                      ;
    STSY        ,A4                             ;
                ,A15  ,'     Plot Mode:'        ;
    STSPM       ,A5                             ;
                ,A17  ,'     Action Mode:';
    STSAM       ,A9                             ;
    ESC2        ,A1                             ;
                ,A2   ,'[H'                     ;
RECORD CLRSTS                                   ;
    ESC3        ,A1                             ;
                ,A6   ,'[24;1H'                 ;
    ESC4        ,A1                             ;
                ,A2   ,'[J'                     ;
RECORD                                          ;
    KEYTYP      ,D3                             ;
    KEY         ,D3                             ;
    FUNCT       ,D1   ,2                        ;
RECORD                                          ;
```

Graphics sketchpad (cont.)

```
    CANCEL      ,D1   ,8                    ; Cancel Key
    EXIT        ,D2   ,10                   ; Exit Key
    NOISE       ,D2   ,11                   ; Beep on Error
    HELP        ,D2   ,15                   ; Help Key
    DO          ,D2   ,16                   ; Do key
    VECTOR      ,D2   ,17                   ; Vector Plot
    RECT        ,D2   ,18                   ; Rectangle
    CIRCLE      ,D2   ,19                   ; Circle
    PRINT       ,D2   ,20                   ; Print Screen
    TRACE       ,D2   ,21                   ; Trace Mode
    WRITE       ,D2   ,22                   ; Write Mode
    ERASE       ,D2   ,23                   ; Erase mode
    SETHOM      ,D2   ,24                   ; Set home
    HOMCUR      ,D2   ,25                   ; Home cursor
    CLRSCR      ,D2   ,26                   ; Clear screen
    UP          ,D2   ,27                   ; Move Cursor Up
    LEFT        ,D2   ,28                   ; Move Cursor Left
    DOWN        ,D2   ,29                   ; Move Cursor Down
    RIGHT       ,D2   ,30                   ; Move Cursor Right
RECORD                                      ;
    CX          ,D4   ,0500                 ; Current X (Def: 500)
    CY          ,D4   ,0500                 ; Current Y (Def: 500)
    PX          ,D4                         ; Stored X position
    PY          ,D4                         ; Stored Y position
    XX          ,D4                         ; Vector end-point X
    YY          ,D4                         ; Vector end-point Y
    HOMEX       ,D4   ,0500                 ; Home of X (Def:500)
    HomeY       ,D4   ,0500                 ; Home of Y (Def:500)
                                            ;
    VCTMOD      ,D1   ,0                    ; Vector Mode flag
    V1          ,D1                         ; Locator Action flag
    DUMMY       ,D1                         ; Dummy Argument
    WK          ,D1                         ; Funct key parameter
                                            ;
    CURMOD      ,D1   ,4                    ; Writing Mode
    LOUD        ,D1   ,1                    ; Bell on error flag
                                            ;
    MODNAM      ,A5   ,'Write'              ; Mode Name
    ACTNAM      ,A9   ,'Plot     '          ; Action name
                                            ;
PROC                                        ;
                                            ;
    OPEN        (1,O,'TT:')                 ; Open the terminal
    XCALL FLAGS ( '50000' )                 ; Disable echo
    XCALL ASCII ( 27, ESC1 )                ; Get the Escape
    ESC2        = ESC1                      ;
    ESC3        = ESC1                      ;
    ESC4        = ESC1                      ;
    XCALL DCGL ( GIC )                      ; Initialize core
    XCALL DCGL ( GSW,0,1000,0,625)          ; Square
    XCALL DCGL ( GNF )                      ; New frame
    CALL CLEAR                              ; Clear , display Sts
```

Graphics sketchpad (cont.)

```
    XCALL DCGL ( GSWM, CURMOD )         ; Set the writing mode
    XCALL DCGL ( GSLS, 1, 0, 0 )        ; Line style is SOLID
    XCALL DCGL ( GMA2, CX, CY )         ; Move absolute
    XCALL DCGL ( GLA2, CX, CY )         ; Line absolute
;
;
; Main plotting loop
;
;
MAIN,                                   ;
    CALL CMDS                           ; Get movement
    XCALL DCGL ( GLA2, CX, CY )         ; Move absolute
    GOTO      MAIN                      ; And Loop
;
;
; The movement Routine
;
;
CMDS,                                   ;
    XCALL DGETK ( KEYTYP )              ; Get next key
                                        ;
    IF (KEYTYP.GT.FUNCT) GOTO NOKEY     ; If Err - Ring Bell
                                        ;
    IF (KEY.EQ.UP)        CY=CY+YINCR   ; Move Up
    IF (KEY.EQ.UP)        RETURN        ; And return
    IF (KEY.EQ.RIGHT)     CX=CX+XINCR   ; Move right
    IF (KEY.EQ.RIGHT)     RETURN        ; and return
    IF (KEY.EQ.DOWN)      CY=CY-YINCR   ; Move down
    IF (KEY.EQ.DOWN)      RETURN        ; and return
    IF (KEY.EQ.LEFT)      CX=CX-XINCR   ; Move left
    IF (KEY.EQ.LEFT)      RETURN        ; and return
                                        ;
    IF (VCTMOD)           GOTO VCTFIX   ; Vector, go by table
                                        ;
    IF (KEY.EQ.EXIT)    GOTO EXIT       ; Dispatch
    IF (KEY.EQ.NOISE)   GOTO NOISE      ;
    IF (KEY.EQ.HELP)    GOTO HELP       ;
    IF (KEY.EQ.VECTOR)  GOTO VCTPLT     ;
    IF (KEY.EQ.RECT)    GOTO RECT       ;
    IF (KEY.EQ.CIRCLE)  GOTO CIRCLE     ;
    IF (KEY.EQ.PRINT)   GOTO PRINT      ;
    IF (KEY.EQ.TRACE)   GOTO TRACE      ;
    IF (KEY.EQ.WRITE)   GOTO WRITE      ;
    IF (KEY.EQ.ERASE)   GOTO ERASE      ;
    IF (KEY.EQ.SETHOM)  GOTO SETHOM     ;
    IF (KEY.EQ.HOMCUR)  GOTO HOMCUR     ;
    IF (KEY.EQ.CLRSCR)  GOTO CLEAR      ;
;
;
; Unassigned Function Key
;
;
```

Graphics sketchpad (cont.)

```
    DISPLAY (1,CLRSTS)                    ; Clear status line
    DISPLAY
&        (1,'Unassigned Function Key')    ; Display Error
    SLEEP 1                               ; Sleep a second
    CALL      STATUS                      ; Redisplay Status Line
    RETURN                                ; And return
                                          ;
;
;
; Vector Fix
;
; Check function keys in Locator Mode, DO key successful
; termination, Help key unsuccessful (for now)
;
;
VCTFIX,                                   ;
    IF (KEY.EQ.DO)      V1 = 1            ; Success Do
    IF (KEY.EQ.CANCEL) V1 = 0             ; Non-Success Cancel
    IF (KEY.EQ.TRACE) GOTO TRACE          ; Switch to Trace Mode
    IF (KEY.EQ.WRITE) GOTO WRITE          ; Switch to Write Mode
    IF (KEY.EQ.ERASE) GOTO ERASE          ; Switch to Erase Mode
    RETURN                                ;
;
;
; Non Function Key
;
;
NOKEY,                    ;
    IF (LOUD) DISPLAY (1,7)               ; If Loud ring a bell
    RETURN                                ;
;
;
; Move cursor to Home position
;
;
HOMCUR,                                   ;
    CX = HOMEX                            ; New X position
    CY = HOMEY                            ; New Y position
    XCALL DCGL ( GMA2, CX, CY )           ; Move to abs position
    RETURN                                ; And return
;
;
; Print Screen Image without Status Line
;
;
PRINT,                                    ;
    DISPLAY (1,CLRSTS,ESC1,'#7')          ; Clear the Status
    CALL STATUS                           ; Redisplay Status Line
    RETURN                                ; Return
;
;
; Clear Screen
```

Graphics sketchpad (cont.)

```
;
;
CLEAR,                                  ;
    XCALL DCGL ( GEV )                  ; Erase Viewport
    CALL STATUS                         ; Redisplay Status Line
    RETURN                              ; And return
;
;
;   Set writing mode to erase mode and update status line.
;   If in locator mode, don't update real writing mode
;
;
ERASE,                                  ;
    MODNAM    = 'ERASE'                 ; Mode Name
    CURMOD    = 8                       ; Erase Mode
    CALL STATUS                         ; Display Status Line
    IF (VCTMOD) RETURN                  ; Locator Mode Return
    XCALL     DCGL ( GSWM, CURMOD )     ; Set Writing Mode
    RETURN                              ; Return
;
;
; Set mode to replace mode and update status line. In Locator
;   Mode, don't update real writing mode
;
;
WRITE,                                  ;
    MODNAM    = 'WRITE'                 ; Mode Name
    CURMOD    = 4                       ; Crnt Mode = Overlay
    CALL STATUS                         ; Update Status Line
    IF (VCTMOD) RETURN                  ; Return (Locator Mode)
    XCALL DCGL ( GSWM, CURMOD )         ; Set writing Mode
    RETURN                              ; Return
;
;
; Trace Mode - no writing, just movement. Do not alter actual
; writing mode in Locator mode.
;
;
TRACE,                                  ;
    MODNAM    = 'TRACE'                 ; Mode Name
    CURMOD    = 0                       ; Mode is TRANSPARENT
    CALL STATUS                         ; Redisplay Status line
    IF (VCTMOD) RETURN                  ; Return (Locator Mode)
    XCALL DCGL ( GSWM, CURMOD )         ; Set Writing Mode
;
;
; Toggle beep on bad input
;
;
```

Graphics sketchpad (cont.)

```
NOISE,                                          ;
    LOUD       = LOUD - 1                       ; Decrement Loud
    IF         (LOUD.EQ.-1) LOUD = 1            ; If neg, turn it on
    RETURN                                      ; Return
;
;
; Home is where the cursor is. Update status line
;
;
SETHOM,                                         ;
    HOMEX      = CX                             ; Update X
    HOMEY      = CY                             ; Update Y
    CALL STATUS                                 ; Display status line
    RETURN                                      ; And return
;
;
; Help the User. No help yet
;
;
HELP,                                           ;
    DISPLAY
&   (1,CLRSTS,'HELP NOT AVAILABLE.')            ; Write Error
    SLEEP 2                                     ; Sleep a couple
    CALL STATUS                                 ; Redisplay Status line
    RETURN                                      ; And return
;
;
; Plot a vector from here to located point.
; Leave cursor at end of Vector.
;
;
VCTPLT,                                         ;
    ACTNAM     = 'VECTOR    '                   ; Action name is VECTOR
    CALL STATUS                                 ; Redisplay the Status
    PX         = CX                             ; Save Crnt position
    PY         = CY                             ;
    CALL VCTFND                                 ; Find the end point
    IF (V1.EQ.0) GOTO RSTCRS                    ; Restore if cancelled
    XX         = PX                             ; Prepare vector
    YY         = PY                             ;
    CALL DRWVEC                                 ; Draw the vector
    ACTNAM     = 'Plot      '                   ; Back into Plot Mode
    CALL STATUS                                 ; Redisplay status line
    RETURN                                      ; And return
;
;
; Draw a rectangle by finding the opposite corner
;
;
```

Graphics sketchpad (cont.)

```
RECT,                                   ;
    ACTNAM = 'Rectangle'                ; Action Name
    CALL STATUS                         ; Update Status line
    PX        = CX                      ;
    PY        = CY                      ;
    CALL      VCTFND                    ; Find the end point
    IF (V1.EQ.0) GOTO RSTCRS            ; Restore if Cancelled
    XCALL DCGL ( GMA2, PX, PY )         ; Put corner back
    XCALL DCGL ( GRA2, CX, CY )         ; Draw the rectangle
    GOTO RSTCRS                         ; Restore the cursor
;
;
; Draw a circle with center here and radius located
; Operation can be cancelled
;
;
CIRCLE,                                 ;
    ACTNAM = 'Circle   '                ; Circle option
    CALL STATUS                         ; Update the Status
    PX        = CX                      ; Hang on to the start
    PY        = CY                      ; point for center
    CALL VCTFND                         ; Locate a radius
    IF (V1.EQ.0) GOTO RSTCRS            ; Quit if cancelled
    XCALL DCGL ( GMA2, CX, CY )         ; Back to center
    XCALL DCGL ( GAA2,PX,PY,CIRCUM)     ; Draw the circle
                                        ; Fall into RstCrs
;
;
; Restore the cursor after locator find
;
;

RSTCRS,                                 ;
    CX        = PX                      ; Reset crnt position
    CY        = PY                      ;
    XCALL DCGL ( GMA2, CX, CY )         ; Restore the cursor
    ACTNAM = 'Plot     '                ; Back to Plot mode
    CALL STATUS                         ; Update status
    RETURN                              ; And return
;
;
; Locator - Find endpoints of vector with one end here
;
;
VCTFND,                                 ;
    XCALL DCGL ( GSWM, 2 )              ; Mode to COMPLEMENT
    VCTMOD = 1                          ; We are in vector mode
    V1 = 0                              ;
    XX = CX                             ;
    YY = CY                             ;
    CALL DRWVEC                         ; Draw the vector
```

Graphics sketchpad (cont.)

```
WHILOP,                                         ;
    IF (V1.NE.0) GOTO NOMORE                    ; We want DO WHILE here
    CALL DRWVEC                                 ; Draw the vector
    XX = CX                                     ;
    YY = CY                                     ;
    CALL DRWVEC                                 ; Draw the vector
    CALL CMDS                                   ; Call commands
    GOTO WHILOP                                 ; Complete While loop
NOMORE,                                         ;
    CALL DRWVEC                                 ; One more time
    XCALL DCGL ( GSWM, CURMOD )                 ; Restore writing mode
    VCTMOD = 0                                  ; No more vector mode
    RETURN                                      ; All done
;
;
; Draw vector
;
; This routine plots a vector from point (Px,Py) to point
; (Xx,Yy) in ; whatever writing mode
;
;
DRWVEC,                                         ;
    XCALL DCGL ( GMA2, PX, PY )                 ; Start point
    XCALL DCGL ( GLA2, XX, YY )                 ; Small vector
    RETURN                                      ; And return
;
;
; Display a status line at the bottom of the screen and put
; the cursor back at the top of the screen
;
;
STATUS,                                         ;
    STSX = HOMEX                                ; The cursor position
    STSY = HOMEY                                ;
    STSPM = MODNAM                              ; Plot Mode:
    STSAM = ACTNAM                              ; Action Mode:
    DISPLAY (1,STATS)                           ; Print status line
    RETURN                                      ; And return
;
;
; Exit the program
;
;
EXIT,                                           ;
    CALL CLEAR                                  ; Clear the screen
    DISPLAY (1,CLRSTS)                          ; Clear the status line
    XCALL DCGL ( GTC )                          ; Terminate Graphics
    STOP                                        ; Bye, Bye
                                                ;
    END                                         ;
```

APPENDIX A

CHARACTER COLLATING SEQUENCE

This appendix contains two tables. Table A-1 shows the English, German, Dutch, and French collating sequences; and Table A-2 shows the multi-national collating sequence.

English, German, Dutch, and French Collating Sequences

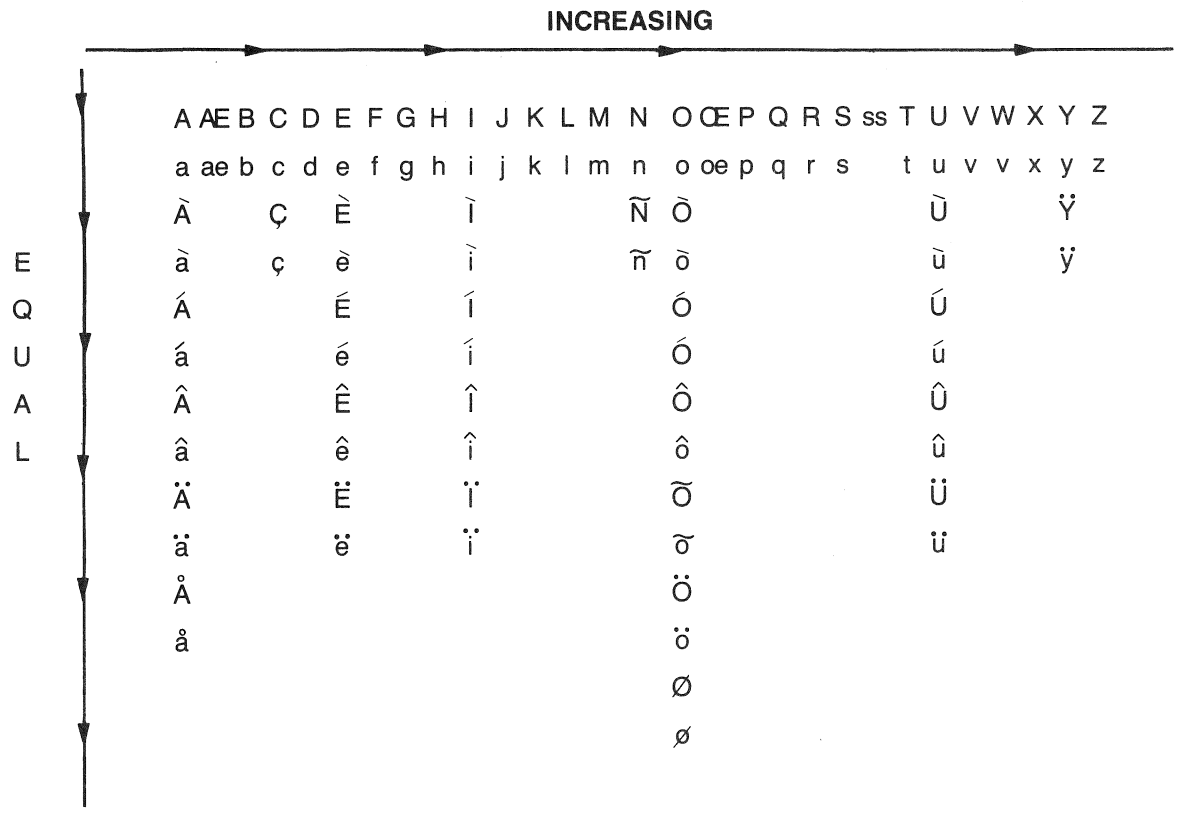

Table A-2

Multi-National Collating Sequence

INCREASING

EQUAL	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	Œ	O	P	Q	R	S	ss	T	U	V	W	X	Y	Z	Æ	Ø
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	ñ	oe	o	p	q	r	s		t	u	v	v	x	y	z	ae	ø
	À		Ç		È				Ì								Ò							Ù				Ÿ			
E	à		ç		è				ì								ò							ù				ÿ			
Q	Á				É				Í								Ó							Ú							
U	á				é				í								ó							ú							
A	Â				Ê				Î								Ô							Û							
L	â				ê				î								ô							û							
	Ä				Ë				Ï								Õ							Ü							
	ä				ë				ï								õ							ü							
	Å																Ö														
	å																ö														

APPENDIX B

This appendix contains two tables. Table B-1 contains the ASCII character set. Table B-1 contains the multi-national character set.

TABLE B-1

ASCII Character Set

Decimal Code	Character	Decimal Code	Character	Decimal Code	Character
000	CTRL/@ (NUL)	042	*	085	U
001	CTRL/A	043	+	086	V
002	CTRL/B	044	,	087	W
003	CTRL/C	045	-	088	X
004	CTRL/D	046	.	089	Y
005	CTRL/E	047	/	090	Z
006	CTRL/F	048	0	091	[
007	BELL (CTRL/G)	049	1	092	\
008	CTRL/H	050	2	093	]
009	HT (Horizontal Tab)	051	3	094	^ {↑}
010	LF (Line Feed)	052	4	095	_ {←}
011	VT (Vertical Tab)	053	5	096	`
012	FF (Form Feed)	054	6	097	a
013	CR (Carriage Return)	055	7	098	b
014	CTRL/N	056	8	099	c
015	CTRL/O	057	9	100	d
016	CTRL/P	058	:	101	e
017	CTRL/Q	059	;	102	f
018	CTRL/R	060	<	103	g
019	CTRL/S	061	=	104	h
020	CTRL/T	062	>	105	i
021	CTRL/U	063	?	106	j
022	CTRL/V	064	@	107	k
023	CTRL/W	065	A	108	l
024	CTRL/X	066	B	109	m
025	CTRL/Y	067	C	110	n
026	CTRL/Z (logical	068	D	111	o
	end-of-file)	069	E	112	p -0
027	ESC (ALTMODE)	070	F	113	q -1
028	CTRL/\	071	G	114	r -2
029	CTRL/]	072	H	115	s -3
030	CTRL/^	073	I	116	t -4
031	CTRL/_	074	J	117	u -5
032	SPACE	075	K	118	v -6
033	!	076	L	119	w -7
034	"	077	M	120	x -8
035	#	078	N	121	y -9
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL (RUBOUT)

TABLE B-2

Multi-National Character Set

Decimal Code	Character	Decimal Code	Character	Decimal Code	Character
128		171	<<	214	Ö
129		172		215	Œ
130		173		216	Ø
131		174		217	Ù
132	IND	175		218	Ú
133	NEL	176	°	219	Û
134	SSA	177	±	220	Ü
135	ESA	178	²	221	Ÿ
136	HTS	179	³	222	
137	HTJ	180		223	ß
138	VTS	181	µ	224	à
139	PLD	182	¶	225	á
140	PLU	183	·	226	â
141	RI	184		227	ã
142	SS2	185	¹	228	ä
143	SS3	186	º	229	å
144	DCS	187	>>	230	æ
145	PU1	188	¼	231	ç
146	PU2	189	½	232	è
147	STS	190		233	é
148	CCH	191	¿	234	ê
149	MW	192	À	235	ë
150	SPA	193	Á	236	ì
151	EPA	194	Â	237	í
152		195	Ã	238	î
153		196	Ä	239	ï
154		197	Å	240	
155	CSI	198	Æ	241	ñ
156	ST	199	Ç	242	ò
157	OSC	200	È	243	ó
158	PM	201	É	244	ô
159	APC	202	Ê	245	õ
160		203	Ë	246	ö
161	¡	204	Ì	247	œ
162	¢	205	Í	248	ø
163	£	206	Î	249	ù
164		207	Ï	250	ú
165	¥	208		251	û
166		209	Ñ	252	ü
167	§	210	Ò	253	ÿ
168	¤	211	Ó	254	
169	©	212	Ô	255	
170	ª	213	Õ		

APPENDIX C

DIBOL OSSL EXTERNAL SUBROUTINES

This appendix describes the external subroutines that are supplied with Tool Kit DIBOL in the OSSL (Operating System Specific Library, DBPOSL.OLB). The subroutines in the UESL (Universal External Subroutine Library, DBPLIB.OLB) are described in the **DIBOL-83 Language Reference Manual** and provide a variety of functions that are commonly required by DIBOL users. All the subroutines described in this appendix can be linked with any DIBOL program using the application build procedures described in Chapter 4 of this manual; they are accessed with the DIBOL XCALL statement.

Chapters 7, 8, and 9 contain, respectively, procedures for accessing FMS, P/OS callable images and system services, and the CORE Graphics Library.

The subroutines documented in this appendix are:

FUNLK	TBENG
ISMCRE	TBFRE
PRINT	TBGER
R5ASC	TBMOD
SYSID	TBMUL
TBBIN	TBRET
TBDUT	TBRPL

C.1 FUNLK

The FUNLK external subroutine unlocks a locked (or not locked) file.

```
XCALL FUNLK (filespec[,status])
```

filespec — The file specification of the file(s) to unlock. Wildcarding is permissable. The defaults are:

Device:	SY0:
Directory:	current directory
Filename:	no default
Extension:	.DDF
Version:	;*

status — Optional decimal variable to which is returned the number of files unlocked.

C.2 ISMCRE

The ISMCRE external subroutine provides the DIBOL program with the ability to create RMS ISAM files. The files that are created may have multiple keys (not segmented keys), initial block allocations, programmer defined bucket sizes, and a specified protection code.

```
XCALL ISMCRE (  File_Name                ; File Name
              ,Max_Rec_Size              ; Maximum Record Size
              ,Posn_Arr                  ; Array of Key Start Posn
              ,Len_Arr                   ; Array of Key Lengths
              ,[Dupl_Arr]                ; Array of Duplicate Flags
              ,[Key_Chg_Arr]             ; Array of Change Flags
              ,[Number_Keys]             ; Number of keys defined
              ,[Init_Alloc]              ; Initial Allocation
              ,[Bucket_Size]             ; Size per bucket
              ,[Protection]    )         ; Protection Code
```

File Information:

File_Name — An alpha record, field, or literal specifying the name of the file to create. The default extension is '.ISM'.

Max_Rec_Size — A decimal expression equal to the length of the longest record that the file will contain. RMS fixed length records are created.

Init_Alloc — An optional decimal expression equal to the initial allocation of the file in blocks. If this argument is not passed, or a null (,,) argument is passed, no initial allocation is assumed.

Bucket_Size — An optional decimal expression equal to the bucket size, in disk blocks, that you want RMS to use for this indexed file. If this argument is not supplied, RMS calculates a bucket size big enough to contain at least one record.

Protection — An optional expression representing the protection code to be given to the file when created. The protection code is expressed similar to the call for 'FLAGS'. A value of zero or space indicates the corresponding protection be denied while a non-zero, non-space value would grant the particular access to the file.

The file protection value identifies the file access privileges of four classes of users:

System	Owner	Group	World

Each of the categories contain four types of access to be granted or denied:

Read	Write	Extend	Delete

Therefore, a 16 character field is required to express the protection code for a particular file. If the field is shorter than sixteen characters, then the omitted classes and access types are assumed to be zero, or denied.

Key Information:

All key information is passed in arrays with the Number_of_Keys argument indicating the number of valid entries in each of the arrays. In order to create an ISAM file with three keys, for example, each of the arrays would be declared as decimal arrays with three entries.

By default each array contains only one entry which allows you to pass literal numbers in place of the arrays when creating an ISAM file with only one key. If there is to be more than one key in a file, it is imperative that you declare these arrays in the data area of the program. Arrays with more than one entry cannot be expressed as literals from DIBOL.

Posn_Arr — A decimal array containing the starting positions of each of the keys.

NOTE

The first character in the record is position ONE, which is not the way it is expressed in the RMS utility RMSDEF.

Len_Arr — A decimal array containing the lengths, in characters, of each of the keys.

Dupl_Arr — A decimal array containing a flag that indicates whether this particular key may have duplicates. A value of zero or space means 'No Duplicate Keys'. If defining only one key and this argument is omitted, then a default value of 'No Duplicate Keys' is assumed.

Key_Chg_Arr — A decimal array containing a flag that indicates whether a key may be changed during an update operation. Note that this parameter affects only ALTERNATE keys and not the PRIMARY key. This argument must be present in the array for the primary key, but is ignored for that key. If the value of this expression is zero or a space, the key may not be changed during an UPDATE operation.

Examples:

1. Create an ISAM file named TEST.ISM with the following:

   a. KEY #1 - Start:4, Length:6, Duplicates:No
   b. The maximum record size is 80 characters

```
XCALL  ISMCRE  ('Test',80,4,6)
```

2. Create an ISAM file named TEST.ISM with the following:

   a. Four keys

```
   Key #1 - Start:3,   Length:6,  Dupl:No
   Key #2 - Start:12,  Length:4,  Dupl:Yes, Change:Yes
Key #3 - Start:80,  Length:6,  Dupl:Yes, Change:No
   Key #4 - Start:255, Length:12, Dupl:Yes, Change:No
```

   b. Maximum Record Length of 512
   c. Bucket size of 4 blocks
   d. Initial allocation of 250 blocks
   e. Protection of (System:RWED,Owner:RWED,Group:,World:)

```
Record
     Pos       ,4D3, 3,12,80,255   ; Keys starting positions
     Len       ,4D2, 6, 4, 6, 12   ; Key lengths
     Dupl      ,4D1, 0, 1, 1,  1   ; Duplicate Key Flags
     Chng      ,4D1, 0, 1, 0,  0   ; Allow UPDATE change flags

PROC

XCALL ISMCRE ('Test',512,Pos,Len,Dupl,Chng,4,250,4,'11111111')
```

C.3 PRINT

The PRINT external subroutine enables a DIBOL program to access all the functions available through the print services task which is documented in the **Professional Tool Kit User's Guide.**

The calling sequence is in the form:

```
XCALL PRINT (status,code[,arg(s)])
```

where:

status — is a decimal array with two entries. The values returned for the two entries are documented in Chapter 6 of the **Professional Tool Kit User's Guide.**

code — is a decimal variable or literal indicating the print call function. The codes are:

Code	Function
1	Print File or Files
2	Abandon Print Job
3	Pause Print Job
4	Continue Print Job
5	Restart Current File
6	View Status

arg(s) — is an optional argument (or arguments) that are used only with "code 1".

The form for "code 1" is:

```
XCALL PRINT (status, 1, filename(s) [,#files])
```

where:

filename(s) — is an alpha literal, or variable when printing one file, or an alpha array when printing a group of files.

#files — an optional decimal expression indicating the number of file names in the alpha array. If only one file is being printed, this argument may be omitted and a value of 1 is assumed.

The non-trappable error #6 (Incorrect Number of Args) is generated if the correct number of arguments (either 1, 2, or 3) are not specified. Error #58 (Program Startup Error) is generated if the print utility cannot be started. Error #104 (Value out of Range) is generated if the code is not in the range 1 through 6.

Example:

This program accepts, from the keyboard, up to nine file names to be printed, and reports the ststus.

```
          Record

          Status    ,2D3       ; Decimal array passed for status
          AStat     ,2A3       ; Used to Display Status
          Files     ,9A50      ; File Names to Print
          NumFil    ,D1        ; Number of Files in 'Files'
          Alpha     ,A1        ; Used to do READS of 'Code'
          Code      ,D1        ; Decimal for code 1 thru 6

     Proc
          Open      (1,0,'TT:')
     Top, Display   (1,13,10,'Input Function:')
          Reads     (1,Alpha,End)
          Code      = Alpha
          Goto      (Code1,Code2,Code3,Code4,Code5,Code6), Code
          Display   (1,13,10,'Range 1 to 6')
          Goto      Top

     Code1, Display (1,13,10,'How many files?')
          Reads     (1,Alpha,Code1)
          Numfil    = Alpha
          If        ((NumFil.Gt.0).And.(NumFil.LE.9)) Goto Okay
          Display   (1,13,10,'Range 1 to 9')
          Goto      Code1
     Okay,          For Code  From 1 Thru NumFil
                    Begin
                         Display (1,13,10,'File Name? ')
                         Reads (1,Files(Code))
                    End
          XCall     Print (Status,1,Files,NumFil)
          Call      STS
          Goto      Top

     Code2,
     Code3,
     Code4,
     Code5,
     Code6,         XCall     Print (Status,Code)
          Call           STS
          Goto           Top

     Sts, AStat(1)  = Status(1)
          AStat(2)  = Status(2)

          Display   (1,13,10,'Status returned = ',
     &               AStat(1),',',Astat(2))
          Display   (1,13,10,'Press RESUME to continue')
          XCall     DWTRE
          Return
```

C.4 R5ASC

The R5ASC external subroutine changes one RAD50 word to three ASCII characters.

The calling sequence is in the form:

```
XCALL R5ASC (afield1,afield2)
```

where:

afield1 is the data field consisting of two alpha characters (one RAD50 word).

afield2 is the data field consisting of three alpha characters. This is the destination field for the converted RAD50 word.

The non-trappable error #6 (Incorrect Number of Args) is generated if two arguments are not specified.

The non-trappable error #31 (Argument Wrong Size) is generated if the size of afield1 is not two or more characters.

C.5 SYSID

The SYSID subroutine returns the CPU identification number.

The calling sequence is in the form:

```
XCALL SYSID (dfield)
```

where:

dfield — is a decimal variable which receives the CPU identification number. The recommended size of this variable is 16.

The error message #6 (Incorrect Number of Args) is generated if other than one argument is specified.

C.6 TBBIN

The TBBIN external subroutine replaces the previously selected collating sequence used for alpha comparisons with a collating sequence consisting of ASCII values for binary comparison.

NOTE

This binary comparison is the default collating sequence for DIBOL.

The calling sequence is in the form:

```
XCALL TBBIN
```

There are no arguments.

C.7 TBDUT

The TBDUT external subroutine replaces the previously selected collating sequence table used for alpha comparisons with a table containing the Dutch collating sequence. Both sequences are established by DEC Standard 169. Appendix A contains tables showing character collation sequences.

The calling sequence is in the form:

```
XCALL TBDUT
```

There are no arguments.

C.8 TBENG

The TBENG external subroutine replaces the previously selected collating sequence table used for alpha comparisons with a table containing the English collating sequence. Both sequences are established by DEC Standard 169. Appendix A contains tables showing character collation sequences.

The calling sequence is in the form:

```
XCALL TBENG
```

There are no arguments.

C.9 TBFRE

The TBFRE external subroutine replaces the previously selected collating sequence table used for alpha comparisons with a table containing the French collating sequence. Both sequences are established by DEC Standard 169. Appendix A contains tables showing character collation sequences.

The calling sequence is in the form:

```
XCALL TBFRE
```

There are no arguments.

C.10 TBGER

The TBGER external subroutine replaces the previously selected collating sequence table used for alpha comparisons with a table containing the German collating sequence. Both sequences are established by DEC Standard 169. Appendix A contains tables showing character collation sequences.

The calling sequence is in the form:

```
XCALL TBGER
```

There are no arguments.

NOTE

Proper names are a special case in the German language. They are not collated in the same way as the rest of the language. The TBGER subroutine does not make this distinction.

C.11 TBMOD

The TBMOD external subroutine permits changing one or more of the entries in the current collating table used for alpha comparisons.

The calling sequence is in the form:

```
XCALL TBMOD (offset1,value1[offset2,value2,...offsetn,valuen])
```

where:

offsetn — is a decimal field or literal specifying an offset into the collating table.

valuen — is a decimal field or literal specifying the new contents of the entry selected by the paired "offset" value.

The error message #6 (Incorrect Number of Args) is generated if the arguments are not paired or if there are no arguments specified.

The error message #104 (Value out of Range) is generated if the value of any argument does not fall within the range of 0 to 255 inclusive.

Example:

```
RECORD
    OFFSET,   D3
    NEWVAL,   D3
    COUNT,    D3
PROC
        OPEN (1,O,'TT:')
        XCALL DECML ('a',OFFSET)        ; offsets start at letter 'a'
        XCALL DECML ('A',NEWVAL)        ; values start with ASCII
LOOP,   XCALL TBMOD (OFFSET,NEWVAL)     ; for 'A'
        INCR COUNT
        INCR OFFSET
        INCR NEWVAL
        IF (COUNT.LT.26) GOTO LOOP      ; repeat 26 times
        IF ('ABCDE'.EQ.'abcde') DISPLAY (1.13.10,'SUCCESS!!')
        IF ('ABCDE'.NE.'abcde') DISPLAY (1.13.10,'FAILURE??')
        CLOSE 1
        STOP
```

This program will print on the terminal:

```
SUCCESS!!
```

C.12 TBMUL

The TBMUL external subroutine replaces the previously selected collating sequence table used for alpha comparisons with a table containing the multi-national collating sequence. Both sequences are established by DEC Standard 169. Appendix A contains tables showing character collation sequences.

The calling sequence is in the form:

```
XCALL TBMUL
```

There are no arguments.

C.13 TBRET

The TBRET external subroutine returns one or more of the entries in the current collating table.

The calling sequence is in the form:

```
XCALL TBRET (offset1,value1[offset2,value2,...offsetn,valuen])
```

where:

offsetn — is a decimal field or literal specifying an offset into the collating table.

valuen — is a decimal field into which is placed the decimal value of the contents of the entry selected by the paired "offset" value.

The error message #6 (Incorrect Number of Args) is generated if the arguments are not paired or if there are no arguments specified.

The error message #104 (Value out of Range) is generated if the value of the "offset" argument does not fall within the range of 0 to 255 inclusive.

C.14 TBRPL

The TBRPL external subroutine replaces the current collating table that is used for alpha comparisons with a user-developed table.

The calling sequence is in the form:

```
XCALL TBRPL (table)
```

where:

table — is a record containing any number of D3 fields. If there are n D3 fields, the first n entries in the table are replaced.

NOTE

The first D3 field corresponds to the first entry (the 0th entry) in the collating table.

- If an entry in the replacement record is zero, the current collating table is unchanged for that entry position.
- If an entry in the replacement table is non-zero, the current collating table is changed to the specified value for that position.
- If the replacement table is smaller than the current collating table, the values in the replacement table are used (according to the rules above) on a one for one basis starting with the first character position. Characters in the current collating table that do not correspond with any in the replacement table are unchanged.

The error message #6 (Incorrect Number of Args) is generated if other than one argument is specified.

The non-trappable error message #31 (Argument Wrong Size) is generated if the record size is not divisible by three.

INDEX

A

Addressing, terminal, 2-4
Application Builder (PAB), 1-2, 4-1
  DIBOL programs, 4-7
  files used by, 4-11
  operating characteristics with, 2-10
  switches, 4-10
Application development, major steps in, 2-2 to 2-5, Figure 2-1
Application Diskette Builder, 2-11,
Application installation file, 1-3, 2-10
Application messages, 1-3
  builder error, 4-8
Application programs,
  developed on host system, 1-1
  installed on the Professional, 1-3
  transfer from host to Professional, 2-11
ASCII characters,
  8-bit instead of 7-bit, 2-4
  table, B-1

B

/B, compiler switch, 3-5
Block mode operation, 5-10
Blocks, virtual, 5-9
Breakpoint control,
  clearing, 6-5
  iteration of, 6-6
  setting, 6-5
Bucket, 5-8
  locking, RMS-11 uses, 5-8
  records cannot cross boundaries, 5-17
Building a DIBOL program for CORE Graphics, 9-2

C

.CMD files,
  application builder files, 3-1
  command files, 4-9
.CMD files (continued)
  file specification section, 4-9
Callable immages/routines, 8-1
Callable P/OS system services, 8-21 to 8-23
Callable system routines, 8-24 to 8-28
Calling CORE Graphics library subprograms, 9-1
Characters,
  display, 2-4
  8-bit, 2-4
  natural (human) language, 2-4
Clustered libraries,
  Professional supports, 1-3
  with .CMD files, 4-4
Code,
  creation with any available editor, 2-10
  non-requirment on Professional, 1-3
Collating sequence, A-1
  English, German, Dutch, and French, A-2
  Multi-national, A-3
Command files,
  editing, 4-4
  generating, 4-3
  to control application builder, 3-1
Command interface (ProDispatcher), 1-4
Command language, host system (DCL, MCR), 1-2
Command qualifiers, compiler, 3-4
Commands,
  application builder format, 4-9
  DDT, 6-2
  global definition, 4-4
Communications facility, 2-6
Compiler, 3-1
  command qualifiers, 3-4
  example, 3-9
  DIBOL programs compiled with, 2-10
  invoked using host command format, 3-2
  output files, 3-6
Compiling with DDT, 6-1

Condition handling, error messages, 2-8
Constant declaration file (CORE Graphics), 9-3
Control instructions (CORE Graphics), 9-4 to 9-6
CTRL/G with DDT, 6-3
CTRL/Z with DDT, 6-3
Current position and marker instructions (CORE Graphics), 9-14 to 9-17

D

/D, compiler switch, 3-5, 6-1
Data storage space requirements, 5-15
DCL command language, 1-2, 3-4
DDT, 6-1
  commands, 6-2
  execution under control of, 2-11
  object module, 1-1
  on the Professional, 2-8
  program execution, 6-3
Debugging, on the Professional, 1-1, 2-3
DELETE statement, 5-5
Descriptor files,
  editing overlay, 4-7
  generating, 4-3
Device codes, Professional system does not require, 1-3
Device drivers, ProDispatcher provides access to, 1-3
DIBOL Debugging Technique (DDT), see DDT
DIBOL ISAM utility not supported, 2-8
DIBOL system language differences, 1-3, 1-4
DIBOL programs,
  application building, 4-7
  development, 2-1
  for the Professional, 1-1, 2-1
  migration to Professional, 2-11
  overall process for, 2-1 to 2-4
  system considerations for, 2-4
DIBOL resident libraries, 4-2
DIBOL Tool Kit Software, 1-1
  compiler, 1-1
  DDT object module, 1-1
  OSSL libraries, 1-1
  run-time system modules, 1-1
  UESL libraries, 1-1
Diskettes, producing customer-usable package, 2-11
Display characters, 2-4
  8-bit instead of 7-bit ASCII, 2-4
DMS files, not supported on Professional, 2-8
Documentation, 1-2
  primary sources of, 2-3

E

Editor (PROSE), 2-6
EFNs, see Event flags
8-bit characters, access natural (human) language characters, 2-4
End-user interface, characteristics of Professional, 2-4
Error messages,
  condition handling, 2-8
  DIBOL compiler, 3-8
  DDT, 6-2
  RMS-11, 5-12
Escape sequences,
  VT100, 2-5
  peculiar to the Professional, 2-5
Event flags (EFNs), 4-5
Executable files, transfer of, 1-3, 2-11
External subroutines for the Professional, APPENDIX C

F

Fatal error routine, 8-15
FDT (frame development tool), 2-5
File access,
  block mode, 5-10
  indexed, 2-7, 5-6
  relative, 2-7. 5-6
  RMS, 2-6, 5-5
  sequential, 2-7, 5-5

File (RMS) services, 2-6
Files,
  DIBOL ISAM not supported on Professional, 2-8
  generating command and descriptor files, 4-3
  map, 4-12
  multivolume not supported on Professional, 2-7
  organizations, Tables 5-1 and 5-2
  overlay descriptor files, 3-1, 4-3, 4-7
  reference to library, 4-11
  task, 4-12
  transfer of executable, 2-11
  used by application builder, 4-11
FMS-11, PRO, 2-5
  argument data types, 7-1
  calling syntax, 7-2
  interface to, 7-1
Form driver, 7-1
Format,
  PAB command, 4-9
  record, 5-7
Frame development tool (FDT), 2-5
FUNLK external subroutine, C-2

G

Get keystroke routine, 8-16
Global attribute instructions (CORE Graphics), 9-10 to 9-13
Global definition commands, 4-4, 4-6
Graphics routines, PRO/GRAPHICS, 2-5
Graphics supplied with the Professional Tool kit, 4-2

H

Help frames, 1-3, 2-5
Help service routines, 8-12 to 8-13
Host environment, program development on, 1-2
Host software, supplied with DIBOL, 1-2
Host system,
  application programs developed on, 1-1
  command language, 1-2

I

Indexed file,
  created using ISMCRE subroutine, 5-2
  created using RMSDEF utility, 5-2, 5-12
  keys, 5-3
  operations, 5-10
  organization, 5-2, 5-17, Figures 5-1 and 5-2
Installing with P/OS, 2-11
Interface to CORE Graphics, 9-1
ISAM utility, not supported on Professional, 2-8
ISMCRE external subroutine, C-3

K

Keyboard function keys, 2-5
Keys,
  alternate, 5-4
  defining, 5-4
  duplicate, 5-4
  indexed file, 5-3, Figures 5-1 and 5-2
  keyboard function, 2-5

L

Label table, compiler, 3-8
Libraries,
  with application builder on host, 4-2
  clustered, Professional supports, 1-3, 4-4
  object libraries for use on host, 2-7
  RMS file access through resident, 4-2

Libraries (continued)
  segmented, Professional supports, 4-2
Library,
  files, 4-11
  resident files, 4-11
Line instructions (CORE Graphics), 9-18 to 9-25
Link with DDT, 6-1
Listing,
  compiler, 3-7
  program, 3-7
  suppress program listing, 3-6
Logical unit,
  CTAB requirements, 4-4
  DIBOL requires first 18, 4-4
  numbers, 4-4
  specified during application building, 2-10
LPQUE statement, 2-6

M

Macros, RMS-11, 5-10
Map file, 4-12
MCR command language, 1-2
Miscellaneous services (P/OS), 8-15
Messages,
  application builder error, 4-8
  compiler error, 3-8
  DDT error, 6-2
  Menus, P/OS feature, 1-3
  condition handling, 2-8
  P/OS message facility 2-7
  RMS-11 error, 5-12
  Suppress DICOMP warning with /W, 3-6
Menu service routines, 8-2 to 8-11
Message service routine, 8-14
Multivolume files, not supported on Professional, 2-8
Multi-national characters, B-1, B-3

N

/N compiler switch, 3-6
Natural (human) language characters, 2-4
New file routine, 8-17

O

.OBJ (Object Module Files), 4-10
.ODL files,
  application builder files, 3-1
  examples of, 4-11
  overlay description language, 4-10
/O, compiler switch, 3-4
O:R submode, 5-6
Object libraries, 2-7
  modules from combined with main programs, 4-1
Object module files, 4-11
Old file routine, 8-18
Operating system functionality, DIBOL differences caused by, 1-4
Operating System Specific Library (OSSL), see Subroutine
Options, compiler qualifiers, 3-4
OSSL, see Subroutine
Overlay descriptor files, 3-1, 4-7, 4-12
Overlay user subroutines, 4-8

P

/P compiler switch, 3-6
Parse string routine, 8-16
P/OS, 1-3
  features of, 1-3
  services and facilities, 2-5
  single user multitask operating system, 1-3
P/OS services and facilities, 2-5
  callable services, 8-21 to 8-23
  Communications Facility, 2-6
  FDT (frame development tool), 2-5
  Messages, 2-7
  PRO/FMS-11, 2-5
  PRO/GRAPHICS, 2-5
  PRO/SORT, 2-6, 8-23
  PROSE (editor), 2-6, 8-21

P/OS Services (continued)
  RMS (file services), 2-6
PAB (Professional Application Builder), 1-2
PRINT external subroutine, C-6
Printing as callable service, 2-6
  use of DIBOL PRINT subroutine, 8-22
PRO/SORT (system service), 8-23
PRODIR (system routine), 8-24
PROFBI (system routine), 8-25
Professional system,
  DIBOL Tool Kit supplied on, 1-1
  Professional Tool Kit supplied on, 1-1
  software environment, 1-3
Professional system services,
  interface routines to, 1-4
Professional Tool Kit, 1-1
  supplied on Professional systems, 1-1
Program,
  development tasks, 2-8
  DIBOL development, 1-1, 2-1
  migration before modification, 1-2
  movement to the Professional, 2-11
PROLOG (system routine), 8-25
PROSE text editor (system service), 8-21
PROVOL (system routine), 8-26

R

R5ASC external subroutine, C-8
Random access,
  mode, 5-6
  statements, 5-5
READ statement, 5-5
READS statement, 5-5
Record,
  cannot cross bucket boundaries, 5-17
  formats, lengths, and storage, 5-6
  operations, 5-9, 5-10
Relative file,
  operations, 5-9
  organization, 5-2, 5-17
Resident libraries,
  DIBOL, 4-2
  files, 4-11
  RMS file access via, 1-3
RMS-11,
  error messages, 5-12
  Macros, 5-10
  record operations, Table 5-2
RMS-11 and DIBOL, 5-1
  access modes, 5-1, 5-5
  creating, 5-6
  file organization, 5-1
  file services, 2-6
RMSDEF utility,
  dialogue, 5-12
  indexed files created using, 5-2, 5-6
RMS file access, 1-3
  support for, 1-3
  via resident libraries, 1-3
RMS file services, 2-6
RMS resident library, symbol table, 4-2
Run-time interpreter, libraries for, 1-3
Run-time language support, in clustered libraries, 2-7

S

/S, compiler switch, 3-6
Send message to message/status display (routine), 8-20
Segmented (RMS) libraries,
  Professional supports, 1-3
Sequential access,
  mode, 5-5
  statements, 5-5
Sequential file,
  operations, 5-9
  organization, 5-2, 5-17
Software,
  DIBOL Tool Kit, 1-1
  Host supplied with DIBOL, 1-2
  Professional environment, 1-3

Sort (PRO/SORT),
  general purpose, 2-6
  similar to SORT-11, 2-6
Source code files, compiler
  accepts, 3-1
Source program, creation of, 2-1,
  2-10
Space requirements, data storage,
  5-17
STORE statement, 5-5
Subroutines,
  linked from object libraries, 1-4
  OSSL library, 4-2
  OSSL and UESL libraries contain,
    2-7
  traceback, 6-9
  UESL library, 4-2
Symbol Table, compiler, 3-6
SYSID external subroutine, C-9
System considerations for DIBOL
  programmers, 2-4
System environment, 1-2
System services, 8-1
  GETKEY, 5-2
System utilities, 4-2

T

Task,
  building a DIBOL task for FMS,
    7-5
  file, 4-12
  executable program, 4-1
TBBIN external subroutine, C-10
TBDUT external subroutine, C-11
TBENG external subroutine, C-12
TBFRE external subroutine, C-13
TBGER external subroutine, C-14
TBMOD external subroutine, C-15
TBMUL external subroutine, C-16
TBRET external subroutine, C-17
TBRPL external subroutine, C-18
Terminal,
  addressing, 2-4
  connected to printer port, DDT,
    6-2
  emulator, 1-2
Terminal driver, ProDispatcher
  provides access to, 1-4
Text editor (PROSE), 2-6
Text instructions (CORE Graphics),
  9-25 to 9-30
TI:, Professional terminal
  identification, 2-4
TT:, terminal identification, 2-4

U

UESL, see Subroutine
Universal External Subroutine
  Library (UESL), see Subroutine

V

Variables,
  examining, 6-8
  manipulation of, 6-7
  setting, 6-7
Viewing transformation instructions
  (CORE Graphics), 9-7 to 9-9
Virtual blocks, 5-9

W

/W, compiler switch, 3-6
Wait for resume key (routine), 8-19
WRITE statement, 5-5
WRITES statement, 5-5

X

XCALL, used with FMS interface, 7-1

Professional Tool Kit
DIBOL User's Guide
Order No. AA-P043C-TK
October 1983

Please cut along this line.

READER'S COMMENTS

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

Did you find errors in this manual? If so, specify by page.

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

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

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
Applied Commercial Engineering MK1-2/H32
Continental Boulevard
Merrimack N.H. 03054

ATTN: Documentation Supervisor

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line