VAXELN Host System Guide

Order Number: AA–JG87B–TE

This manual explains how to use the VAXELN host system to develop VAXELN applications.

Revision/Update Information: This manual supersedes *VAXELN Host System Guide*, AA–JG87A–TE.

Operating System and Version: VAX/VMS, Version 4.0 or later

Software Version: VAXELN, Version 3.0

digital equipment corporation
maynard, massachusetts

First Printing, December 1986
Revised, October 1987

ML-S718

This document was prepared using VAX DOCUMENT, Version 1.0.

Contents

Preface

Manual Objectives

This manual explains how to use the VAXELN host system to develop VAXELN programs and to develop, load, boot, and debug VAXELN system images.

Intended Audience

This manual is for programmers and students who have a working knowledge of the Pascal, C, or FORTRAN programming language. A cursory understanding of the VAX/VMS DIGITAL command language (DCL) is also necessary.

Document Structure

This manual is organized as follows:

- Chapter 1, Host System Overview, provides an overview of the host system's hardware and software and briefly describes the steps in developing a VAXELN application.
- Chapter 2, Program Development, explains how to prepare VAXELN Pascal, VAX C, and VAX FORTRAN programs for inclusion in VAXELN systems. The chapter explains how to compile source files, link object modules and shareable images, and maintain object module and shareable image libraries.

- Chapter 3, System Development, explains how to use the VAXELN System Builder to build VAXELN system images.
- Chapter 4, Booting and Down-Line Loading, describes the procedures for booting and loading a VAXELN system image on a target machine. Procedures for booting and loading by disk and by down-line loading using the Ethernet are explained.
- Chapter 5, Debugging VAXELN Systems, explains how to use the local and remote debuggers provided with the VAXELN development system to debug VAXELN system images.
- Chapter 6, Performance Analysis, explains how to use the VAXELN Performance Utility to collect and report performance data for VAXELN systems.
- Appendix A, VAX–11/750 Microcode Patch, describes the microcode control store patching procedure required on system power-up before you can run VAXELN on a VAX–11/750.
- Appendix B, Using the Error Log Server, explains how to create error log files on a remote node.
- Appendix C, A Full System Map, gives an example of a full map generated by the System Builder.

Conventions

Convention	Meaning
{ }	Braces enclose lists from which you must choose one item. For example: $\left\{ \begin{array}{l} \textit{expression} \\ \textit{statement} \end{array} \right\}$
. . .	Horizontal ellipsis points mean that the item preceding the ellipsis points can be repeated. For example: `/qualifier . . .`
. . .	Vertical ellipsis points in a figure or example indicate that not all the information the system would display is shown or that not all the information a user is to supply is shown.

Convention	Meaning
{ }, . . .	Braces followed by a comma and horizontal ellipsis points mean that you can repeat the enclosed item one or more times, separating two or more items with commas.
{ }; . . .	Braces followed by a semicolon and horizontal ellipsis points mean that you can repeat the enclosed item one or more times, separating two or more items with semicolons.
[ ]	Square brackets mean that the statement syntax requires the square bracket characters. This notation is used with Pascal arrays, sets, and attribute lists. For example: `ARRAY[INDEX1]` Square brackets are also used in the syntax of a directory name in a VAX/VMS file specification and in user identification code (UIC) specifications.
⟦ ⟧	Double brackets enclose optional items. For example: ⟦*/qualifier* . . . ⟧
UPPERCASE characters	VAX/VMS, VAXELN, and language-specific reserved words and identifiers are printed in uppercase characters. However, you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase characters.
lowercase characters	Elements you must replace according to the description in the text are printed in italic lowercase characters. However, you can enter them in lowercase, uppercase, or a combination of lowercase and uppercase characters.
RETURN	In examples, carriage returns are implied at the end of each line. However, the RETURN symbol is used in some examples to emphasize that you must press the Return key.
CTRL/x	CTRL/x indicates a control key sequence. Press the key labeled CTRL while you simultaneously press another key. For example: CTRL/C
user input	In interactive examples, **bold** print indicates user input.
decimal notation	Numeric values are represented in decimal notation unless otherwise noted.

Associated Documents

The following documents are part of the VAXELN document set:

- *VAXELN Release Notes*
- *Introduction to VAXELN*
- *VAXELN Installation Manual*
- *VAXELN Run-Time Facilities Guide*
- *VAXELN Run-Time Utilities Guide*
- *VAXELN Application Design Guide*
- *VAXELN Pascal Language Reference Manual: Language Elements*
- *VAXELN Pascal Language Reference Manual: Real-Time Programming Routines*
- *VAXELN Pascal Language Reference Manual: Device Drivers, Program Development*
- *VAXELN C Run-Time Library Reference Manual*
- *VAXELN FORTRAN Programmer's Guide*
- *VAXELN Master Index*
- *VAXELN Messages Manual*

The following documents are relevant to VAXELN:

- *VAX/VMS DCL Dictionary*
- *VAX/VMS Linker Reference Manual*
- *VAX/VMS Library Reference Manual*
- *VAX C User's Guide*
- *VAX/VMS Error Log Utility Reference Manual*
- *VAX/VMS System Manager's Reference Manual*
- *Guide to Creating Modular Procedures on VAX/VMS*
- *Guide to VAX/VMS System Management and Daily Operations*
- *DECnet DIGITAL Network Architecture General Description*
- *DECnet–VAX System Manager's Guide*
- *DECnet–VAX User's Guide*
- *MicroVAX I Owner's Manual*
- *MicroVAX II Owner's Manual*

- *LSI–11 Analog System User's Guide*
- *DLV11–J User's Guide*

Chapter 1

Host System Overview

VAXELN is a software tool kit that provides a high-performance, run-time system for real-time and/or distributed applications. VAXELN lets you build VAX applications for use in configurations for which VAX/VMS might be an inappropriate operating system. Some VAXELN application areas include robotics, process control, image processing, seismic data collection, patient monitoring, test equipment automation, and simulation.

The VAXELN tool kit consists of host development software and target system software. The host development system is a VAX/VMS or MicroVMS system on which you can design, develop, and debug VAXELN applications. This chapter provides an overview of host system hardware and software and explains how to use the host system to develop a VAXELN application.

1.1 Hardware

VAXELN supports a variety of devices and processors for VAXELN system development and execution. To develop a VAXELN system, you need a VAX processor running the VAX/VMS or MicroVMS operating system. If you want to use the remote debugger, you also need a DECnet license.

1.2 Software

In addition to VAX/VMS text editors and other utilities, the VAXELN host system software includes the following development utilities:

- **VAXELN Pascal Compiler** — The VAXELN Pascal compiler builds object modules from Pascal source code. VAXELN Pascal contains extensions for systems programming. For more information about this compiler, see the *VAXELN Pascal Language Reference Manual.*
- **System Builder** — The VAXELN System Builder builds a system image by combining application program images with the kernel, run-time libraries, the File Service, the Network Service, device drivers, terminal drivers, and debuggers. Chapter 3 provides a complete description of the System Builder.
- **Debuggers** — VAXELN provides a remote debugger that lets you debug a system image from the host development system, using the Ethernet. In addition, the tool kit includes a local debugger that allows you to debug VAXELN systems on the target. For more information about VAXELN debuggers, see Chapter 5.

1.3 Developing a VAXELN Application

This section provides an overview of the steps in developing a VAXELN application with the host system. The DCL commands you can use for each step are illustrated in the following example. Later chapters provide detailed explanations for each step.

```
❶ $ EPASCAL MYFILE1/DEBUG+ELN$:RTLOBJECT/LIBRARY
  $ EPASCAL MYFILE2/DEBUG+ELN$:RTLOBJECT/LIBRARY
  $ LINK MYFILE1+MYFILE2+ELN$:RTLSHARE/LIBRARY+RTL/LIBRARY
❷ $ EBUILD MYSYSTEM
❸ $ EDEBUG/LOAD=MYSYSTEM.SYS VAXNODE
```

To create a VAXELN application, you must:

❶ Develop the programs you want to include in your VAXELN system image by using a high-level language, such as VAXELN Pascal, VAX C, VAX FORTRAN, or VAXELN Ada® . You must create and compile each source program, and link the object modules. The preceding

® Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

example shows you how to compile and link the VAXELN Pascal source files MYFILE1 and MYFILE2.

❷ Use the System Builder to create a system image. You must invoke the System Builder with the EBUILD command and edit the System Builder menus. The preceding example shows you how to invoke the System Builder to create the system image MYSYSTEM.

❸ Boot the system image on the target machine from local mass storage or down-line load the system image on the target machine by using the Ethernet. The preceding example illustrates how to use the VAXELN remote debugger to down-line load a system image. You can then use the remote debugger to debug the system image.

Chapter 2

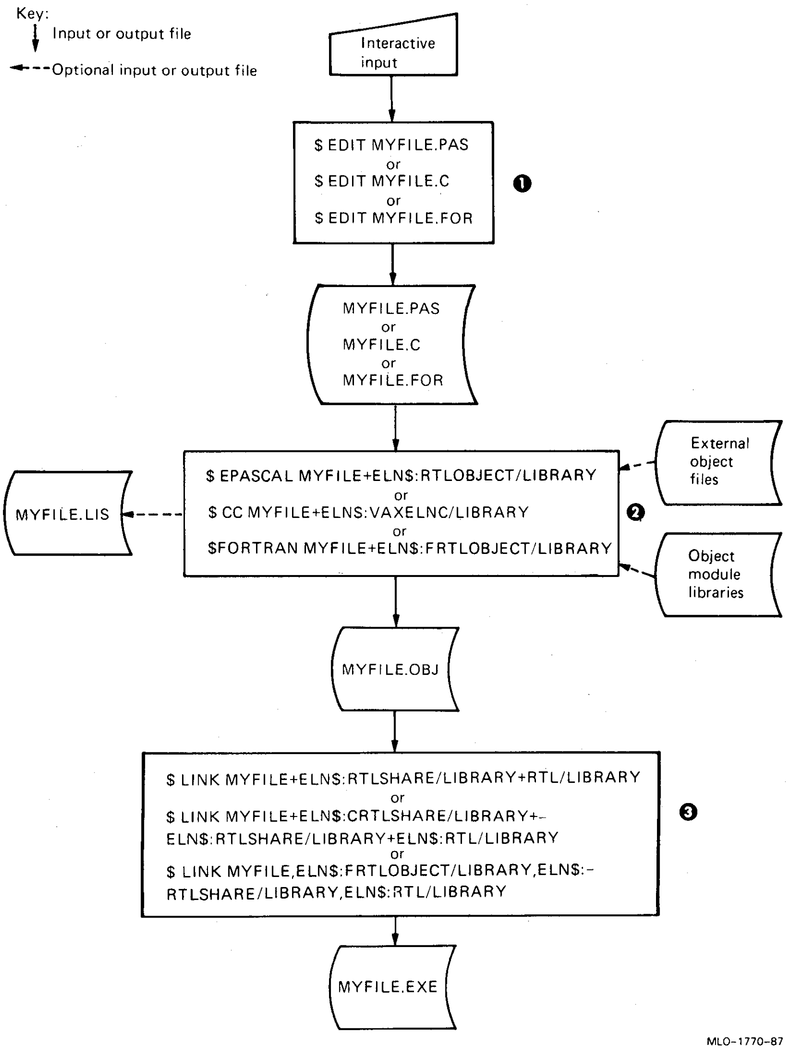
Program Development

This chapter explains how to develop program images for a VAXELN system. The chapter introduces the VAX/VMS DCL commands you can use to create and compile source programs, to link object modules, and to maintain module libraries. After you develop your program images, you can use the VAXELN System Builder to combine the images into a VAXELN system (see Chapter 3).

2.1 Commands for Program Development

Figure 2–1 illustrates the sequence of commands used to develop the program MYFILE. These commands are described in the following sections. The commands in the figure are shown in their simplest forms. You can also specify qualifiers with the commands to request special processing or to indicate a special type of input file.

Figure 2–1: Commands for Program Development



Key:
Input or output file
Optional input or output file
Interactive input
$ EDIT MYFILE.PAS
or
$ EDIT MYFILE.C
or
$ EDIT MYFILE.FOR
❶
MYFILE.PAS
or
MYFILE.C
or
MYFILE.FOR
$ EPASCAL MYFILE+ELN$:RTLOBJECT/LIBRARY
or
$ CC MYFILE+ELNS:VAXELNC/LIBRARY
or
$FORTRAN MYFILE+ELN$:FRTLOBJECT/LIBRARY
❷
MYFILE.LIS
External object files
Object module libraries
MYFILE.OBJ
$ LINK MYFILE+ELN$:RTLSHARE/LIBRARY+RTL/LIBRARY
or
$ LINK MYFILE+ELN$:CRTLSHARE/LIBRARY+-
ELN$:RTLSHARE/LIBRARY+ELN$:RTL/LIBRARY
or
$ LINK MYFILE,ELN$:FRTLOBJECT/LIBRARY,ELN$:-
RTLSHARE/LIBRARY,ELN$:RTL/LIBRARY
❸
MYFILE.EXE
MLO-1770-87

❶ The EDIT command invokes the system editor to create a file containing source code. The file type you should give the source file depends on the language you are using. The file types of source files for the languages used most often to create VAXELN applications are as follows:

Language	**Source File Type**
VAXELN Pascal	PAS
VAX C	C
VAX FORTRAN	FOR

❷ The EPASCAL, CC, or FORTRAN command invokes the appropriate compiler to process the source code and verify that no syntax errors or violations of the language rules exist. The compiler generates an object module of the type OBJ and an optional listing file of the type LIS.

If errors occur, correct them and recompile the source file. If warning or informational messages appear, you can continue. However, if you continue when a warning occurs, you might get unexpected results.

❸ The LINK command invokes the VAX/VMS Linker to combine object modules, shareable images, and the contents of object module and shareable image libraries. In addition, the linker resolves references to symbols, such as routine names. The resulting program image has the file type EXE, and you can include the image in a VAXELN system.

The example in Figure 2–1 assumes that MYFILE.PAS, MYFILE.C, and MYFILE.FOR specify complete programs.

2.2 Compiling Programs

After you create a VAXELN Pascal, VAX C, or VAX FORTRAN source file, compile the file to generate an object module. To compile a VAXELN Pascal source file, use the EPASCAL command; to compile a VAX C source file, use the CC command; to compile a VAX FORTRAN source file, use the FORTRAN command. If you are compiling a source file for the first time, you should specify the /DEBUG qualifier in the command line to include debugging information in the object module. In the following example, the VAXELN Pascal compiler compiles the source file MYFILE.PAS:

```
$ EPASCAL/DEBUG MYFILE
```

This command generates the object module file MYFILE.OBJ.

2.2.1 EPASCAL Command

The syntax for the EPASCAL command and its qualifiers follows:

$ EPASCAL⟦ {*/command-qualifier*} . . . ⟧ *source-file-spec*

⟦ { $\left\{\begin{matrix}+\\,\end{matrix}\right\}$ *file-spec/positional-qualifier*}

⟦ { $\left\{\begin{matrix}+\\,\end{matrix}\right\}$ *file-spec* ⟦ {*/positional-qualifier*} ⟧ } . . . ⟧ ⟧

command-qualifier

A command qualifier, which can appear after the EPASCAL command or after a file specification. In many cases, the default settings for the EPASCAL command are sufficient for compiling source files. However, by specifying the command with qualifiers, you can precisely control the compilation. For example, you can tell the VAXELN Pascal compiler to include debugger information in the object module by specifying the /DEBUG qualifier with the EPASCAL command.

source-file-spec

A standard VAX/VMS file specification that represents the source file to be compiled. If you do not specify a file type, the compiler uses the default file type PAS. If you enter a command without a file specification, the operating system prompts you for a file:

```
$ EPASCAL RETURN
_File:
```

file-spec

A standard VAX/VMS file specification that designates an object library or object module whose exported declarations are to be included during compilation. Libraries have the default file type OLB and object modules have the default file type OBJ. You can specify up to eight object libraries in a command; you can specify any number of object modules. Separate multiple file specifications with commas or plus signs.

The first library and first module file specifications you specify must be followed by the /LIBRARY and /MODULE qualifiers, respectively. Subsequent files can be listed without the qualifier; a positional qualifier affects the files that follow it up to the next file preceding a positional qualifier. For example:

```
$ EPASCAL PROG+ELN$:RTLOBJECT/LIBRARY+ELN$:RTL+NEWLIB+DEFS/MODULE+BASIC_DEFS
```

This command line includes six file specifications — a source file specification, three object library specifications, and two object module specifications. PROG is the source file being compiled. The /LIBRARY qualifier affects the object libraries specified by ELN$:RTLOBJECT, ELN$:RTL, and NEWLIB. The /MODULE qualifier affects the object modules specified by DEFS and BASIC_DEFS.

positional-qualifier

The qualifier /LIBRARY or /MODULE.

You can specify another name for an object file or listing file by including a file specification with the /OBJECT or /LIST qualifier. For example, the following command generates the listing file MYFILELIST.LIS and the object file MYFILE.OBJ:

```
$ EPASCAL/DEBUG/LIST=MYFILELIST MYFILE
```

The following example compiles the source file MYFILE.PAS and generates the object file MYOBJECTFILE.OBJ:

```
$ EPASCAL/DEBUG/OBJECT=MYOBJECTFILE MYFILE
```

For more information about the EPASCAL command and its qualifiers, see the *VAXELN Pascal Language Reference Manual* or use the HELP facility:

```
$ HELP EPASCAL
```

2.2.2 CC Command

The syntax for the CC command and its qualifiers follows:

$ CC⟦*/qualifier* . . . ⟧ *file-spec-list*⟦*/qualifier* . . . ⟧

/qualifier

Specifies an action to be performed by the compiler or a special input file property. You can specify multiple qualifiers. In many cases, the default settings for the CC command are sufficient for compiling source files. However, by specifying the command with qualifiers, you can precisely control the compilation. For example, you can tell the VAX C compiler to include debugger information in the object module by specifying the /DEBUG qualifier with the CC command.

You can append qualifiers to the CC command or to individual file specifications. If you include a qualifier in a list of files to be concatenated, the qualifier affects all the files in the list.

You can use the /LIBRARY qualifier with one or more input files that are part of a list of files separated by plus signs (+). This qualifier indicates that the file is a text library that can contain source text.

For more information about the CC command and its qualifiers, see the *VAXELN C Run-Time Library Reference Manual* or the *VAX C User's Guide*, or use the HELP facility:

```
$ HELP CC
```

file-spec-list

Specifies one or more source files to be compiled. Separate the file specifications with commas (,) or plus signs (+). If you separate source file specifications with commas, the programs are compiled separately. If you separate source file specifications with plus signs, the files are joined and compiled as one program. If you enter a command without a file specification, the operating system prompts you for a file:

```
$ CC RETURN
_File:
```

The default file type for VAX C source files is C. If a file specification is qualified with /LIBRARY, the default file type is TLB.

You can specify another name for an object file or listing file by including a file specification with the /OBJECT or /LIST qualifier. For example, the following command generates the listing file MYFILELIST.LIS and the object file MYFILE.OBJ:

```
$ CC/DEBUG/LIST=MYFILELIST MYFILE
```

The following example compiles the source file MYFILE.PAS and generates the object file MYOBJECTFILE.OBJ:

```
$ CC/DEBUG/OBJECT=MYOBJECTFILE MYFILE
```

For more information about specifying file specifications with the CC command, see the *VAXELN C Run-Time Library Reference Manual* and the *VAX C User's Guide*.

2.2.3 FORTRAN Command

The syntax for the FORTRAN command and its qualifiers follows:

$ FORTRAN⟦/*qualifier* . . . ⟧ *file-spec-list*⟦/*qualifier* . . . ⟧

/qualifier

Specifies an action to be performed by the compiler or a special input file property. You can specify multiple qualifiers. In many cases, the default settings for the FORTRAN command are sufficient for compiling source files. However, by specifying the command with qualifiers, you can precisely control the compilation. For example, you can tell the VAX FORTRAN compiler to include debugger information in the object module by specifying the /DEBUG qualifier with the FORTRAN command.

NOTE

When you use the /DEBUG qualifier with the FORTRAN command, you should also use the /NOOPTIMIZE qualifier.

You can append qualifiers to the FORTRAN command or to individual file specifications. If you include a qualifier in a list of files to be concatenated, the qualifier affects all the files in the list.

You can use the /LIBRARY qualifier with one or more input files that are part of a list of files separated by plus signs (+). This qualifier indicates that the file is a text library that can contain source text.

For more information about the FORTRAN command and its qualifiers, see the *VAX FORTRAN User's Guide* or use the HELP facility:

```
$ HELP FORTRAN
```

file-spec-list

Specifies one or more source files to be compiled. Separate the file specifications with commas (,) or plus signs (+). If you separate source file specifications with commas, the programs are compiled separately. If you separate source file specifications with plus signs, the files are joined and compiled as one program. If you enter a command without a file specification, the operating system prompts you for a file:

```
$ FORTRAN RET
_File:
```

The default file type for VAX FORTRAN source files is FOR. If a file specification is qualified with /LIBRARY, the default file type is OLB.

You can specify another name for an object file or listing file by including a file specification with the /OBJECT or /LIST qualifier. For example, the following command generates the listing file MYFILELIST.LIS and the object file MYFILE.OBJ:

```
$ FORTRAN/DEBUG/LIST=MYFILELIST MYFILE
```

The following example compiles the source file MYFILE.FOR and generates the object file MYOBJECTFILE.OBJ:

```
$ FORTRAN/DEBUG/OBJECT=MYOBJECTFILE MYFILE
```

For more information about specifying file specifications with the FORTRAN command, see the *VAX FORTRAN User's Guide.*

2.3 Linking Object Modules

The VAX/VMS Linker combines object modules and shareable images to produce program images. The program images are then ready for inclusion in VAXELN systems (see Chapter 3.)

To link object modules, use the LINK command. In the following example, the LINK command combines the object modules MYFILE1.OBJ and MYFILE2.OBJ with references from the object module library RTL.OLB and the shareable image library RTLSHARE.OLB:

```
$ LINK MYFILE1+MYFILE2+ELN$:RTLSHARE/LIBRARY+RTL/LIBRARY
```

This command produces the image file MYFILE.EXE.

2.3.1 LINK Command

The syntax for the LINK command and its qualifiers follows:

$ LINK⟦*/qualifier . . .* ⟧ *file-spec-list*⟦*/qualifier . . .* ⟧

qualifier

Descriptions of LINK command qualifiers you might use for developing VAXELN applications are provided in Table 2–1.

file-spec-list

Specifies one or more files to be linked. Each specification indicates an object module generated by a compiler, an object module library, or a shareable image library. Object module files have the default type OBJ. Object module and shareable image libraries have the default file type OLB and must be specified with the /LIBRARY or /INCLUDE file qualifier.

If the first file specification is a library, you must use the /INCLUDE qualifier to indicate which of its modules to use (see Section 2.3.2).

If you specify multiple file specifications, separate the specifications with commas or plus signs (+). If you enter the LINK command without a file specification, the operating system prompts you for a file:

```
$ LINK RETURN
_File:
```

2.3.2 Using Qualifiers to Control the Linker

In many cases, the default settings for the LINK command are sufficient for linking object modules. However, by specifying the command with qualifiers, you can precisely control the linkage of a program image. For example, you can tell the linker to copy debug symbol table information from the object modules to the program image by specifying the /DEBUG qualifier.

You can append qualifiers to the LINK command or to individual file specifications. You must specify library file specifications with the /LIBRARY or /INCLUDE positional qualifier.

Table 2–1 lists ways you can use qualifiers to control linkage. The following sections explain how to use the qualifiers. For additional information about the LINK command and its qualifiers, see the *VAX/VMS Linker Reference Manual.*

Table 2–1: LINK Command Qualifiers

Qualifier	Usage	Default
/⟦NO⟧ DEBUG	Directs the linker to generate a debug symbol table (DST) and to give the debugger control when the program image is run	/⟦NO⟧DEBUG
/LIBRARY	Specifies a library file that is to be included in the linking operation	
/INCLUDE=(*module-list*)	Specifies a library file and the modules in it which are to be included in the linking operation	
/⟦NO⟧ SHAREABLE	Creates a shareable image	/⟦NO⟧SHAREABLE
/⟦NO⟧ SYSLIB	Suppresses the search through the default system libraries for unresolved symbolic references	SYSLIB
/⟦NO⟧ SYSSHR	Suppresses the search through the VAX/VMS default shareable image library for unresolved symbolic references	SYSSHR

2.3.2.1 Generating Debugger Information (/DEBUG)

When you compile a program, you can include debugging information in the resulting object module by specifying the /DEBUG qualifier. If you choose to include debugging information, you must also include that information in the program image by specifying the /DEBUG qualifier with the LINK command. See Chapter 5 for more information about debugging.

2.3.2.2 Linking Libraries (/LIBRARY)

To include the object modules or shareable images of a library file in the linking operation, specify the library file with the /LIBRARY qualifier. The linker searches the object modules or shareable images in the specified library for definitions needed to resolve references from the VAXELN object modules that you specify. For more information, see the *VAX/VMS Linker Reference Manual.*

2.3.2.3 Linking Library Modules and Images (/INCLUDE)

To include a library file's object modules and shareable images in the linking operation, specify the names of the object modules and shareable images with the /INCLUDE qualifier. You must specify at least one module or image with the qualifier. If you specify a list of module or image names, separate them with commas (,).

You can specify the /INCLUDE qualifier with or without the /LIBRARY qualifier. If you know exactly which modules or images you need to include in your program image, specify the /INCLUDE qualifier without the /LIBRARY qualifier. Specifying /INCLUDE without /LIBRARY optimizes your program because the linker searches for unresolved symbols, such as routine names, only in the specified modules and images. If you specify both qualifiers, the linker first includes the specified modules or images and then searches for unresolved references in all other modules or images in the specified library.

2.3.2.4 Creating a Shareable Image (/SHAREABLE)

A shareable image is a nonexecutable image that you can link into program images. If you have a program unit that is invoked by more than one program, linking the unit as a shareable image provides the following benefits:

- Saves disk space — The executable images to which the shareable image is linked do not physically include the shareable image. The system image has only one copy of the shareable image.
- Simplifies maintenance — If you use transfer vectors and the GSMATCH option, you can modify, recompile, and relink a shareable image without having to relink an executable image that is linked with it. For discussions of transfer vectors and the GSMATCH option, refer to the *VAX/VMS Linker Reference Manual.*

To create a shareable image, use the /SHAREABLE qualifier. You can specify the qualifier in a LINK command that contains input specifications for one or more object modules or shareable images. If you specify the name of an existing shareable image, the image must exist in a shareable image library. The default file type of the resulting shareable image is EXE.

To operate on shareable image libraries, specify the LIBRARY command with the /SHARE qualifier and the shareable image files (see Section 2.4.2.5). For detailed information on creating shareable image libraries, see the *Guide to Creating Modular Procedures on VAX/VMS.*

The System Builder ensures that if you include a shareable image in the final system, the program refers to that image. However, the System Builder looks for shareable images in the VAXELN directory ELN$. To use shareable images from another directory, assign the image's file specification to the logical name ELN$*name*, where *name* is the file name of the image. The System Builder first looks for a translation of ELN$*name*. If the System Builder does not find a translation, it looks for the file in the ELN$ directory.

For example, to instruct the System Builder to find the shareable image MYSHARE.EXE in the directory [MYFILES], use one of the following commands:

```
$ ASSIGN [MYFILES]MYSHARE.EXE ELN$MYSHARE
$ DEFINE ELN$MYSHARE [MYFILES]MYSHARE.EXE
```

Any programs that access a shareable image must be linked with that image. When performing the link operation, you must specify one of the following items on your LINK command:

- The name of the shareable image library containing the symbol table of the shareable image — use the /LIBRARY qualifier to identify a library file.
- A linker options file that contains the name of the shareable image file — use the /SHAREABLE qualifier to identify a shareable image file. A shareable image file must be specified in an options file rather than on the command line, because the linker creates a shareable output image whenever the LINK command line includes the /SHAREABLE qualifier.

NOTE

You cannot specify shareable images as VAXELN Pascal compiler input.

2.3.2.5 Suppressing the Search Through the Default System Libraries (/NOSYSLIB)

The linker searches default VAX/VMS system libraries SYS$LIBRARY: IMAGELIB.OLB and SYS$LIBRARY:STARLET.OLB to resolve symbolic references that remain undefined after specified input and user default libraries are processed. If your input and your user default libraries let the linker resolve symbolic references, specify the /NOSYSLIB qualifier. This qualifier prevents the linker from searching the default system libraries. If you do not use this qualifier, you could get incompatible code included in your program image. A correct VAXELN program image should not contain any unresolved references.

2.3.2.6 Suppressing the Search Through the Default Shareable Image Library (/NOSYSSHR)

To prevent the linker from searching SYS$LIBRARY:IMAGELIB.OLB, the VAX/VMS default shareable image library, for unresolved references, specify the /NOSYSSHR qualifier. See Section 2.3.2.5.

2.3.3 Using VAXELN Libraries

The VAXELN toolkit supplies run-time libraries for linking with VAXELN Pascal, VAX C, and VAX FORTRAN object modules. Table 2–2 lists the run-time libraries with brief descriptions.

Table 2–2: VAXELN Run-Time Libraries

Library	Description
CRTLOBJECT.OLB	Contains object module versions of the VAX C run-time routines
CRTLSHARE.OLB	Contains shareable image versions of the VAX C run-time routines
FILE.OLB	Contains the File Service modules and support routines
FRTLOBJECT.OLB	Contains object modules for VAX FORTRAN programming
RTL.OLB	Contains object modules for run-time routines that have local read/write data and a module defining entry points for kernel procedures

Table 2–2 (Cont.): VAXELN Run-Time Libraries

Library	Description
RTLOBJECT.OLB	Contains object module versions of the VAXELN run-time routines
RTLSHARE.OLB	Contains shareable image versions of the VAXELN run-time routines

The installation procedure places these libraries in the directory ELN$ (see the *VAXELN Installation Manual*). To write your own versions of files in this directory, see Section 3.3.

The organization of run-time support into object module and shareable image libraries lets you specify an image's run-time support in several ways:

- If you link the object modules with a shareable library and the RTL.OLB library, program images can share one copy of the routines that are shareable.
- If you link each program's object modules with the RTLOBJECT.OLB, CRTLOBJECT.OLB, or FRTLOBJECT.OLB library and the RTL.OLB library, program images in the final system can use their own local copies of the run-time routines. You should list the RTLOBJECT, CRTLOBJECT, or FRTLOBJECT library first because it refers to symbols defined in RTL.

NOTE

On occasion, a routine contained in the library RTL.OLB will reference a routine contained in the library RTLOBJECT.OLB. Since the typical LINK command specifies RTLOBJECT/LIB before RTL/LIB, the linker will next search the libraries in SYS$LIBRARY to resolve the reference. If your system happens to have VAX Pascal installed, the reference may be erroneously resolved in the VAX Pascal library PASRTL.EXE, whose routines often have the same names as the VAXELN counterparts.

As a result, the System Builder will fail when it attempts to locate ELN$:PASRTL.EXE, even though the linker did not report any undefined global symbols. To ensure that undefined VAXELN global symbols are reported, use the /NOSYSLIB linker command qualifier. /NOSYSLIB tells the linker not to search in SYS$LIBRARY.

Use a LINK command line like the following to resolve all VAXELN references:

```
$ LINK/NOSYSLIB MYPROG+ELN$:RTLOBJECT/LIB+RTL/LIB+ -
RTLOBJECT/LIB+RTL/LIB
```

- You can build a system that has both local and shareable program images — that is, some program images can keep local copies while some share the same copies.

NOTE

When linking a C program, you cannot combine object modules and shareable images. The libraries you link against must either contain all object modules or all shareable images.

When a program image refers to a shareable image, the System Builder ensures that the necessary shareable images are built into the system.

You should include the RTLSHARE.OLB and RTL.OLB libraries in most link operations. For convenience, you can assign these libraries to logical names. For Pascal, you can define the following logical name:

```
$ DEFINE LNK$LIBRARY ELN$:RTLSHARE.OLB
```

For C, you can define the following logical name:

```
$ DEFINE LNK$LIBRARY ELN$:CRTLSHARE.OLB
```

Since VAX FORTRAN routines are only supplied in object library form, you can define the following logical name for FORTRAN:

```
$ DEFINE LNK$LIBRARY ELN$:FRTLOBJECT.OLB
```

You can use the following definition for the VAXELN run-time object library:

```
$ DEFINE LNK$LIBRARY_1 ELN$:RTL.OLB
```

The preceding commands define LNK$LIBRARY and LNK$LIBRARY_1 to be the default libraries for the linker. When these definitions are in effect, you need only specify the object module or modules for your program. For example:

```
$ LINK MYOBJECT
```

You can display the names of the files in a library by using the LIBRARY command with the /LIST qualifier. For example:

```
$ LIBRARY/LIST RTLSHARE
```

For more information about the VAXELN run-time libraries, see the *VAXELN Run-Time Facilities Guide*. For more information about the LIBRARY command, see Section 2.4.1.

2.3.4 Selecting the Default Double-Precision Type

When the VAXELN toolkit is installed, libraries are configured so that the VAX D_floating format is used by run-time routines for double-precision operations. (Some mathematical run-time routines generate temporary double-precision values even when your program has not declared the data type DOUBLE.)

Two command procedures exist in the directory ELN$ that let you specify the double-precision format for the machine for which you are developing a VAXELN system: GFLOATRTL.COM and DFLOATRTL.COM. GFLOATRTL.COM makes G_floating the default representation for the run-time routines. The following command changes the default to the G_floating representation:

```
$ @ELN$:GFLOATRTL
```

To go back to D_floating as the default representation, use the command:

```
$ @ELN$:DFLOATRTL
```

D_floating is the default double-precision floating-point representation for the VAXELN Pascal and VAX C compilers. To generate G_floating instructions, use the compiler's /G_FLOATING command qualifier. For example, to prepare a program for a MicroVAX computer that has F_floating and G_floating formats, specify the /G_FLOATING qualifier when you compile the program and use the GFLOATRTL.COM command procedure to update the run-time library representation. You can then omit the emulation software (described in Chapter 3) for floating-point instructions.

2.4 Maintaining Libraries

Use the VAX/VMS Librarian to maintain object module and shareable image libraries. To invoke the librarian, use the LIBRARY command. The following command places the object file MYFILE.OBJ in the object module library OBJECTLIB.OLB:

```
$ LIBRARY OBJECTLIB MYFILE
```

For more information about the librarian or about the LIBRARY command, see the *VAX/VMS Library Reference Manual.*

2.4.1 LIBRARY Command

The syntax for the LIBRARY command and its qualifiers follows:

$ LIBRARY⟦/*qualifier* . . . ⟧ *library-file-spec* ⟦*input-file-spec* ⟦, . . . ⟧⟧

/qualifier

Descriptions of the LIBRARY command qualifiers are provided in Table 2–3.

library-file-spec

Specifies the name of an object module library or a shareable image library. Libraries have the default file type OLB.

input-file-spec

With the /CREATE qualifier, you can specify one or more object files to be included in a new library. If you specify more than one input file, separate the file specifications with commas (,).

2.4.2 Using Qualifiers to Control the Librarian

You can control the VAX/VMS Librarian by specifying the LIBRARY command with qualifiers. For example, you can tell the librarian to create a new library by specifying the /CREATE qualifier and a library file specification.

You can append qualifiers to the LIBRARY command or to individual file specifications. When you include a qualifier in a list of files to be concatenated, the qualifier affects all the files in the list.

Table 2–3 lists ways you can use qualifiers to control the librarian. The following sections explain how to use the qualifiers. Additional information about LIBRARY command qualifiers is provided in the *VAX/VMS Library Reference Manual.*

Table 2–3: LIBRARY Command Qualifiers

Qualifier	Default
/CREATE	Creates a new library
/INSERT	Inserts files into an existing library
/LIST⟦=*file-spec*⟧	Lists a library's contents
/EXTRACT=*(module-list)*	Extracts modules from a library
/REPLACE	Replaces existing library modules with new files
/SHARE	Indicates a shareable image library
/DELETE=*(module-list)*	Deletes modules from a library
/COMPRESS	Compresses a library

2.4.2.1 Creating a New Library (/CREATE)

To create a new library, specify the LIBRARY command with the /CREATE qualifier and a library specification. You can also specify a list of object file specifications. If you do, their modules are inserted in the new library. For example:

```
$ LIBRARY/CREATE OBJECTLIB MYFILE1,MYFILE2
```

This command creates the object module library OBJECTLIB.OLB and places the object files MYFILE1.OBJ and MYFILE2.OBJ in that library.

2.4.2.2 Inserting or Replacing Modules in an Existing Library (/INSERT and /REPLACE)

The /INSERT and /REPLACE qualifiers are defaults. If you specify the names of object files that are not already in the library, they are inserted. If you specify the names of object modules that already exist in the library, the old modules are replaced. Thus, you can insert or replace object modules in an existing library by specifying the LIBRARY command as follows:

```
$ LIBRARY OBJECTLIB MYFILE1,MYFILE2
```

2.4.2.3 Listing a Library's Contents (/LIST)

Use the /LIST qualifier to list the contents of a library. For example:

```
$ LIBRARY/LIST RTL
```

This command writes the list of the object modules in the library RTL.OLB to SYS$OUTPUT (usually your terminal).

If you want to write the list of object modules to a file, specify a listing file specification as follows:

```
$ LIBRARY/LIST=RTLMODULES RTL
```

This command writes the object modules in the RTL.OLB library to the file RTLMODULES.LIS.

2.4.2.4 Extracting Modules from a Library (/EXTRACT)

To extract one or more modules from a library, use the /EXTRACT qualifier. For example:

```
$ LIBRARY/EXTRACT=(MODULE1,MODULE2) OBJECTLIB
```

The modules MODULE1.OBJ and MODULE2.OBJ are removed from the library OBJECTLIB.OLB and are placed in the default directory.

If you want to rename the output files, specify the /OUTPUT qualifier with new output file specifications:

```
$ LIBRARY/EXTRACT=(MODULE1,MODULE2)/OUTPUT=(MOD1,MOD2) OBJECTLIB
```

The files MODULE1.OBJ and MODULE2.OBJ are extracted from the library OBJECTLIB.OLB and are renamed MOD1.OBJ and MOD2.OBJ.

2.4.2.5 Creating a Shareable Image Library (/SHARE)

To create a shareable image library, use the /SHARE qualifier with the /CREATE qualifier. For example:

```
$ LIBRARY/CREATE/SHARE MYSHARLIB MYSHRIMG1,MYSHRIMG2
```

This command creates the shareable image library MYSHARLIB.OLB and places the shareable images MYSHRIMG1.EXE and MYSHRIMG2.EXE in that library.

2.4.2.6 Deleting Modules from a Library (/DELETE)

To delete one or more modules from a library, use the /DELETE qualifier. For example:

```
$ LIBRARY/DELETE=(MODULE1,MODULE2) OBJECTLIB
```

This command deletes the object modules MODULE1.OBJ and MODULE2.OBJ from the library OBJECTLIB.OLB.

2.4.2.7 Compressing a Library (/COMPRESS)

You can recover unused space in a library resulting from module deletions by compressing the library. Specify the /COMPRESS qualifier and the name of the library you want to compress:

```
$ LIBRARY/COMPRESS OBJECTLIB
```

By default, a new library file is created that has the same specification as the one compressed, but whose version has increased by one. You can assign a new name to the output file by specifying the /OUTPUT qualifier with an output file specification.

Chapter 3

System Development

After you develop the programs you need to include in your VAXELN system, use the System Builder to create the system.

This chapter explains how to:

- Use the EBUILD command to invoke the System Builder
- Use the EBUILD qualifiers to control the System Builder
- Use your own versions of DIGITAL-supplied VAXELN files
- Edit System Builder menus

3.1 EBUILD Command

The EBUILD command invokes the System Builder to combine one or more program images into a bootable system image. The syntax for the command and its qualifiers follows:

$ EBUILD⟦/*qualifier* . . . ⟧ *data-file-spec*

/qualifier

Descriptions of the EBUILD command qualifiers are provided in Table 3–1.

data-file-spec

Specifies the data file that is to store information about your VAXELN system. The default file type for a data file is DAT. If you are in EDIT mode (the default) and you specify a file that does not exist, the System Builder creates that file. If the file exists, the System Builder reads the file, creates a new version of the file when you make a change, and uses the file's data to set up menus.

If you specify a data file and the /NOEDIT qualifier, the System Builder reads that file and uses its contents to create a VAXELN system image file.

If you enter the EBUILD command without a data file specification, the System Builder prompts you for the file:

```
$ EBUILD [RETURN]
_File:
```

NOTE

> You should create and modify the data file only through the System Builder. The results of editing the data file in any other way are unpredictable.

3.2 Using Qualifiers to Control the System Builder

In many cases, the default settings for the EBUILD command are sufficient for building system images. However, by specifying the command with qualifiers, you can precisely control the System Builder. For example, you can tell the System Builder to generate a system map by specifying the /MAP qualifier.

You can append qualifiers to the EBUILD command or to the data file.

Table 3–1 lists qualifiers you can use to control the System Builder. The following sections explain how to use the qualifiers.

Table 3–1: EBUILD Command Qualifiers

Qualifier	Usage	Default
/⟦NO⟧BRIEF	Generates a brief system map	/BRIEF
/⟦NO⟧EDIT	Generates a system image through EDIT mode	/EDIT
/⟦NO⟧FULL	Generates a full system map	/NOFULL
/⟦NO⟧LOG	Suppresses display of the system image size	/LOG

Table 3–1 (Cont.): EBUILD Command Qualifiers

Qualifier	Usage	Default
/MAP=⟦*file-spec*⟧ /NOMAP	Generates a system map	/NOMAP
/SYSTEM=*file-spec*	Specifies a name for the system image file	The data file name

3.2.1 Generating a System Map (/MAP, /FULL, /BRIEF)

To generate a system map when you build a system image, use the /MAP qualifier. For example, the following command names the system map file MYDATAFILE.MAP:

```
$ EBUILD/MAP MYDATAFILE
```

If you specify the /MAP qualifier with a file specification, the System Builder writes the map output to that file. For example:

```
$ EBUILD/MAP=MYMAPLISTING MYDATAFILE
```

In this case, the System Builder generates the system map file MYMAPLISTING.MAP.

You can control the contents of the system map by using the mutually exclusive qualifiers /BRIEF and /FULL (/NOBRIEF is the same as /FULL). By default, the System Builder generates a brief map that lists the images you include in the system, the devices and terminals you specify, and the system characteristics. A full map lists images you include in the system and their program descriptions, devices and their device descriptions, terminal descriptions, and system characteristics. To generate a full system map, use the following command:

```
$ EBUILD/MAP/FULL MYDATAFILE
```

See Appendix C for an example of a full system map.

3.2.2 Generating a System Image Without Entering EDIT Mode (/NOEDIT)

By default, the System Builder executes in an interactive screen-editing mode that is compatible with VT100- and VT200-series terminals. EDIT mode lets you interactively modify system characteristics and programs. If you want the System Builder to build a system image without entering EDIT mode, specify the /NOEDIT qualifier. This qualifier causes the System Builder to create the system image from the contents of the specified data file. For example:

```
$ EBUILD/NOEDIT MYDATAFILE
```

3.2.3 Suppressing the Display of System Image Size (/NOLOG)

After creating a system image, the System Builder displays the size of that system image at the terminal. You can prevent the System Builder from displaying the image size by using the /NOLOG qualifier. For example:

```
$ EBUILD/NOLOG MYDATAFILE
```

3.2.4 Specifying a Name for the System Image File (/SYSTEM)

The System Builder uses the name of the data file that you specify with the EBUILD command and the SYS file type to name your system image. You can override this naming convention by specifying a file name with the /SYSTEM qualifier. For example:

```
$ EBUILD/SYSTEM=MYSYSTEM MYDATAFILE
```

The System Builder uses the data file MYDATAFILE.DAT to create the system image MYSYSTEM.SYS.

3.3 Using Your Own Versions of VAXELN Files

You can build a system using your own versions of Digital-supplied files ordinarily located in the VAXELN directory ELN$. A system map shows you what files the System Builder looks in ELN$ for. To use your own version of an ELN$ file, you must first assign your file specification the logical name ELN$*name*, where *name* is the name of your file. For example, to input your own CONSOLE.EXE file, use one of the following DCL commands.

```
$ ASSIGN [MYFILES]CONSOLE.EXE ELN$CONSOLE
$ DEFINE ELN$CONSOLE [MYFILES]CONSOLE.EXE
```

You would also need to assign the file specification to ELN$*name* if your file was in the ELN$ directory but under a different name from the file it was replacing, for example, if your file was named MYCONSOLE.EXE.

The System Builder automatically looks for a translation of ELN$CONSOLE. If it finds a translation, the System Builder uses that file instead of ELN$:CONSOLE.EXE. If you do not define ELN$CONSOLE, the System Builder will use ELN$:CONSOLE.EXE.

For a list of files installed in the ELN$ directory, see the *VAXELN Installation Manual.*

3.4 Editing System Builder Menus

When you use EDIT mode, the System Builder displays a series of menus on your terminal screen (VT100- and VT200-series terminals). By editing the menus, you can select the programs, devices, system characteristics, network node characteristics, terminal characteristics, and console characteristics that you want to include in your system image.

To ensure that VAX/VMS has the correct characteristics for your terminal, type the following command before invoking the System Builder:

```
$ SET TERMINAL/INQUIRE
```

To guide your reading, the rest of this chapter will describe creating and editing the data file SAMPLE.DAT. Invoke the System Builder by typing the following command:

```
$ EBUILD SAMPLE
```

The System Builder displays the Main Menu (unless the data file exists and you specify the /NOEDIT qualifier). The name of the system (the data or system file name) on which you are working and the type of menu being displayed appear at the top of each menu. Menu commands appear at the bottom of the menus. Each menu command corresponds to a function key on your terminal's keypad. On the LK201 keyboard, the DO and HELP keys perform their named functions and keys F17 through F20 correspond to keypad keys PF1 through PF4, respectively. Table 3–2 lists the menu commands, their corresponding keys, and brief descriptions.

Table 3–2: Menu Commands

Command	Keypad Key	LK201 Key	Description
BACK	PF4	F20	Displays the previous menu
DELETE	PF3	F19	Deletes a program, device, or terminal description
DO	PF1	DO/F17	Activates the selected entry
EXIT	PF4	F20	Ends the System Builder session and incorporates changes
HELP	PF2	HELP/F18	Displays brief descriptions of the menu entries or the general System Builder interface, depending on the context
QUIT	PF3	F19	Aborts the System Builder without altering the input file (requires confirmation with DO)

To execute a command, select a menu item and press the appropriate key. You can select menu commands, menu entries, or entry options by using the arrow keys (↑, ↓, ←, →). If you are editing a menu for the first time, default entry options are highlighted. Similarly, menu entries and entry options become highlighted when you select them. (If you use a VT100 terminal, you cannot see highlighting unless the terminal has the advanced video option.)

A diamond symbol appearing at the top, bottom, or left or right edge of a menu indicates that additional text exists. To show the additional text, scroll the text by using the arrow keys.

You can also use menu control commands to control the menu displays. Table 3–3 lists the control commands with brief descriptions.

Table 3–3: Menu Control Commands

Command	Description
CTRL/E	Moves the cursor to the end of an argument list
CTRL/H	Moves the cursor to the beginning of an argument list
CTRL/R	Clears the terminal screen
CTRL/U	Deletes text from the cursor back to the beginning of the line

After you edit a menu, use the DO command to incorporate your edits. The System Builder enters the specified characteristics or description into your VAXELN system. To return to the previous menu, use the BACK command (the System Builder does not incorporate your edits). If you use the DO command after entering invalid input for a menu entry, the System Builder gives you a chance to correct the error.

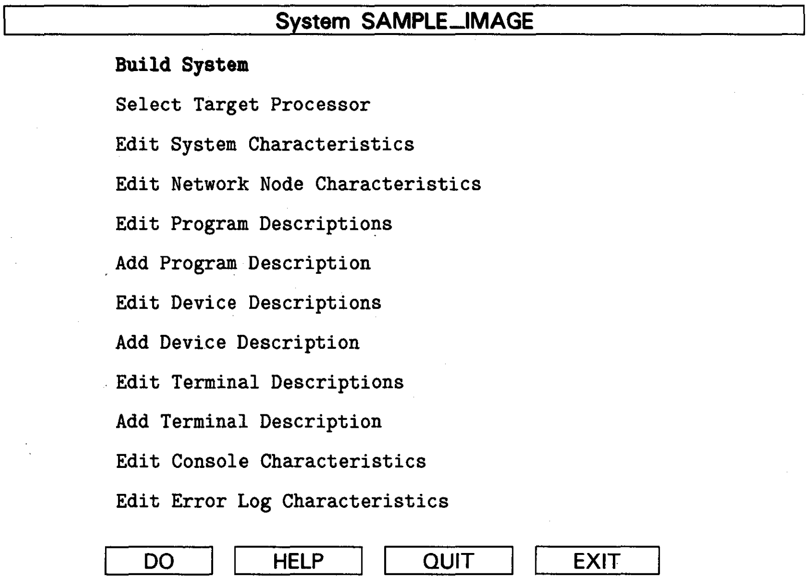
The following sections describe the System Builder menus. Each section includes a figure illustrating menu entries and their defaults and a list of menu entry descriptions. The entries for a menu might not fit on your terminal screen. If the diamond symbol appears at the bottom of your screen, use the Down Arrow key (↓) to scroll the screen and see the rest of the entries.

3.4.1 Main Menu

The Main Menu lists procedures you can use to build a system from information in your data file, to edit the characteristics of the entire system or of the network node, and to add or edit descriptions of programs, devices, terminals, and console characteristics. Figure 3–1 shows this menu.

If you select the **Build System** item on the Main Menu and use the DO command, the System Builder creates a system image file (file type SYS) by combining the information in your data file with the VAXELN kernel and run-time software and exits. If a data file does not exist, the System Builder creates a default data file and uses that file to create the system image file. You should not select **Build System** until after using the Add Program Description Menu.

Figure 3–1: Main Menu



```
System SAMPLE_IMAGE

Build System
Select Target Processor
Edit System Characteristics
Edit Network Node Characteristics
Edit Program Descriptions
Add Program Description
Edit Device Descriptions
Add Device Description
Edit Terminal Descriptions
Add Terminal Description
Edit Console Characteristics
Edit Error Log Characteristics

DO   HELP   QUIT   EXIT
```

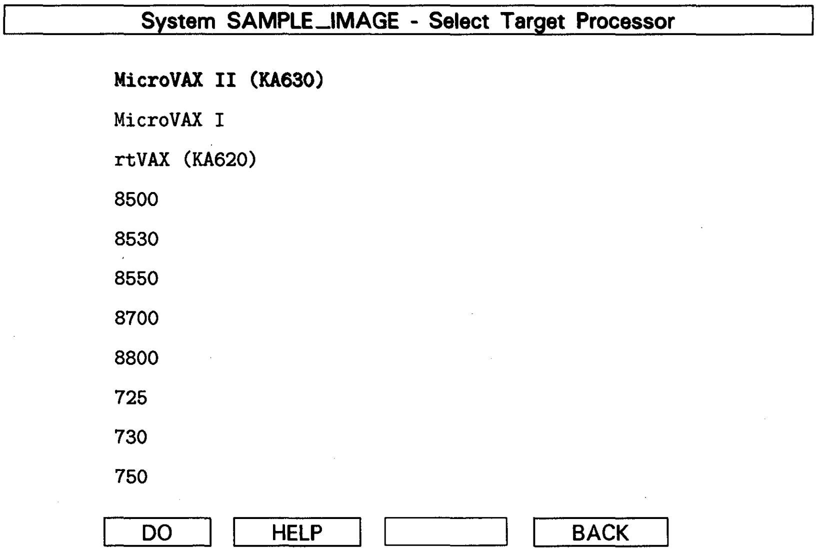
If you select an item other than **Build System** and use the DO command, the System Builder displays another menu.

3.4.2 Editing the Select Target Processor Menu

The **Select Target Processor** entry on the Main Menu displays the Select Target Processor Menu. This menu allows you to specify the target your system will run on. You must specify the target in order to build your system. The default is **MicroVAX II (KA630)**. Figure 3–2 shows the Select Target Processor Menu.

If you are using the VAXStation II, or the VAXStation 2000, select **MicroVAX II** as your target. If you are using an 8800 as a single processor, select **8700**.

Figure 3–2: Select Target Processor Menu



System SAMPLE_IMAGE - Select Target Processor

```
MicroVAX II (KA630)
MicroVAX I
rtVAX (KA620)
8500
8530
8550
8700
8800
725
730
750
```

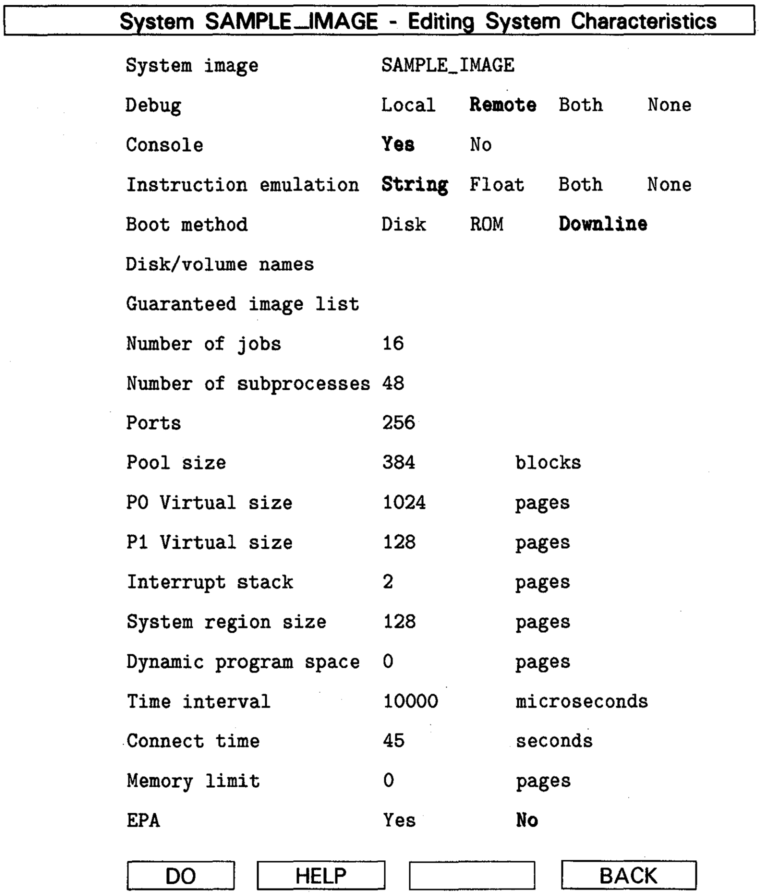
3.4.3 Editing the System Characteristics Menu

If you select the **Edit System Characteristics** entry on the Main Menu, the System Builder displays the System Characteristics Menu, which lists the run-time system characteristics of the system you are building. Figure 3–3 shows this menu.

Descriptions of the entries on the System Characteristics Menu follow:

- **System image** — Specify the name of the system image file. If you specified a file name with the /SYSTEM qualifier in the EBUILD command, the System Builder displays that name here as the default. Otherwise, the System Builder displays the name of your data file with a SYS file type as the default.

Figure 3–3: System Characteristics Menu



System SAMPLE_IMAGE - Editing System Characteristics

System image	SAMPLE_IMAGE			
Debug	Local	**Remote**	Both	None
Console	**Yes**	No		
Instruction emulation	**String**	Float	Both	None
Boot method	Disk	ROM	**Downline**	
Disk/volume names				
Guaranteed image list				
Number of jobs	16			
Number of subprocesses	48			
Ports	256			
Pool size	384	blocks		
P0 Virtual size	1024	pages		
P1 Virtual size	128	pages		
Interrupt stack	2	pages		
System region size	128	pages		
Dynamic program space	0	pages		
Time interval	10000	microseconds		
Connect time	45	seconds		
Memory limit	0	pages		
EPA	Yes	**No**		

- **Debug** — Select **Local** to include the debugger image EDEBUGLCL in the system image. Select **Remote** to include the remote debugger EDEBUGREM in the system image. Select **Both** to include both debuggers. When you include both debuggers, EDEBUGREM is the primary debugger; EDEBUGLCL gains control only if a fatal system error occurs. The default is **Remote**. For more information on the debuggers, see Chapter 5.

NOTE

You might want to include a debugger during the development of a system. You can select a debugger even if no program has the Debug option. The selected debugger gains control if the program does not handle an exception.

- **Console** — Select **Yes** to allow communication with the console terminal on the target machine. The System Builder includes the appropriate console driver and device description. If you select **Local** or **Both** for the Debug option, the console driver and device description are automatically included. The name for the system's console terminal is CONSOLE:. If you select **No** with the debugging option **Remote**, the remote VAX/VMS terminal behaves as the console terminal, when running the remote debugger. The default is **Yes**.
- **Instruction emulation** — This entry selects emulation software for instructions present in the full VAX architecture but not present in the MicroVAX architecture. Select **None** unless you are building a system for a MicroVAX target machine. If your target machine is a MicroVAX I, MicroVAX II, or KA620, you must select **String** to include emulation software for string instructions in the system. **Float** is optional. If you select **Float**, the System Builder includes emulation software for the extended-precision floating-point instructions. Selecting **Both** includes emulation software for both groups of instructions. The default is **String**.

  You can choose the double-precision format in VAXELN Pascal and C programs by using the compiler qualifier /NOG_FLOATING, and you can choose the default double-precision format of the run-time library with the command procedures DFLOATRTL.COM and GFLOATRTL.COM. By using these qualifiers and command procedures, you can omit the slower floating-point emulation software.
- **Boot method** — This entry indicates the method by which the system is to be booted on the target machine. Select **Disk** if the system is to be loaded and booted from a disk or tape device. Select **ROM** if the system is to be loaded and booted from read-only memory (ROM). Select **Downline** if the system is to be down-line loaded. The default is **Downline**.

  The **Boot method** characteristic specifies one of the following to be the type of image header used in the system:

Disk	No header
ROM	MicroVAX ROM header
Downline	DECnet MOP image header

If you specify that a system containing the Network Service is to be booted from a disk, a tape device, or from ROM, but you have not specified a node address, the System Builder displays a warning message.

- **Disk/volume names** — Supply device specifications and volume names in the following format for volumes present on the target machine:

  device-specification volume-nam

  For example:

  ```
  DQA1 TEST1
  ```

  Programs can refer to a volume by prefixing the given name with DISK$. For example, in Pascal:

  ```
  OPEN(FILE1,FILE_NAME := 'DISK$TEST1:[RT.SRC]RXDRIVER.PAS')
  ```

  Separate multiple quoted arguments with commas. The first specification identifies the default volume. The File Service mounts the indicated volumes when the system is booted. The volume name is optional; if you omit it or specify a volume not present in the drive, the File Service attempts to mount the volume that is present in the drive.

  An alternative to specifying volume names on this menu is to use the MOUNT_VOLUME procedure, which also mounts the volume present in the drive, if you omit the volume name.

- **Guaranteed image list** — Specify a list of shareable images that are referred to by programs loaded by the dynamic program loader. Shareable images that are referred to by programs included in the system by the System Builder are included automatically. But shareable images referred to by programs that are not loaded until run time would not otherwise be included.

- **Number of jobs** — Specify an integer in the range 1 to 32767 to indicate the maximum number of jobs that can exist in the system at any one time. The number includes both user and system jobs, for example, device drivers. This value is used to determine the number of P0 page table slots reserved in the system. The default is 16.

- **Number of subprocesses** — Specify an integer in the range 1 to 32767 to indicate the maximum number of subprocesses that can exist in the system at any one time, including both user and system subprocesses. This value plus the number of jobs is used to calculate the number of P1 page table slots to be reserved in the system. The default is 48.

  NOTE

  Overestimating the number of jobs or subprocesses introduces negligible overhead into the system. For each job specified, two bits are allocated in a system bitmap; for each subprocess, one bit is allocated.

- **Ports** — Specify an integer in the range 2 to 32767 to indicate the maximum number of message ports the system can use simultaneously. The default is 256.
- **Pool size** — Specify an integer in the range 16 to 32764 to indicate the number of blocks necessary to store the kernel objects the system uses simultaneously. Each kernel object in use requires one block, processes require three blocks, and each job requires a few additional blocks. Pool blocks are 128 bytes long. The default is 384.
- **P0 Virtual size** — Specify an integer in the range 128 to 32640 to indicate the maximum number of 512-byte pages that each P0 region in the system occupies. The kernel uses this value to allocate page tables for each job. The default is 1024. If the value supplied is not a multiple of 128, the System Builder will round it up to the next multiple of 128.

  Demands on P0 memory arise from the following sources:

  - Program images (variables at the outer level and all code)
  - Heap data and other dynamically allocated memory
  - Created and received messages
  - Accessed AREAs
  - Accessed FORTRAN Commons

  All P0 pages that are set up for a job at build time appear under the job's program information entry on a full EBUILD map. These P0 pages include any demand zero pages that are not stored in the system image on disk. Also, the Pascal NEW procedure dynamically adds to the heap.

- **P1 Virtual size** — Specify an integer in the range 128 to 32640 to indicate the maximum number of 512-byte pages that each P1 region in the system occupies. The kernel uses this value to allocate page tables for each process created in a job. The default is 128. If the value supplied is not a multiple of 128, the System Builder will round it up to the next multiple of 128.

  Demands on P1 memory arise from the following sources:

  - The user stack
  - The kernel stack
  - The process-context page

  All process-local variables (those declared inside a block) are placed on the user or kernel stack in P1 space, depending on which mode the process is running in. Variables declared within the Pascal PROGRAM block and the C main or first function are process-local variables for the master process of a job. Furthermore, all call frames are placed on the appropriate stack.

NOTE

The P0 and P1 virtual size parameters are used as the maximums for all jobs and processes in the system. Therefore, if any one job or process requires a large virtual P0 or P1 size, every job or process in the system will have that amount as its maximum virtual memory size. On rtVAX systems, this can cause a problem.

Because rtVAX systems require that all memory for page table entries be contiguous in physical memory, overestimating the P0 or P1 virtual size can create much greater overhead than on other systems. When each job's page table is set up, memory for page table entries is allocated all at one time and cannot be expanded dynamically. Therefore, when each job is created, sufficient physical memory is allocated for the page table entries for both the P0 and P1 virtual sizes specified on the menu. When each subprocess is created, sufficient physical memory is allocated for the P1 virtual size specified on the menu. If more virtual memory is specified than is needed, memory will be wasted to provide page-table entries for that unused space.

Overestimating the P0 or P1 virtual size on non-rtVAX systems causes little overhead, since physical memory for page table entries is allocated dynamically. If memory within the virtual space specified is not required, no memory for page table entries for that space will be allocated.

- **Interrupt stack** — Specify an integer in the range 2 to 8192 to indicate the maximum number of pages required for the system's interrupt stack. The default is 2.
- **System region size** — Specify an integer in the range 0 to 32767 to indicate the maximum number of pages required by interrupt service communication regions and, if selected, the performance analyzer and error-logging buffers. The kernel uses this value to allocate page table entries for the system during the system's start-up. The default is 128.
- **Dynamic program space** — Specify an integer in the range 0 to 32767 to indicate the number of memory pages the kernel can allocate for loading programs dynamically into the running system. The default is 0.
- **Time interval** — Specify an integer in the range 1 to 120000000 (microseconds) to indicate the interval between interval timer interrupts. The value you specify is the minimum time that can be used for time-dependent operations. Each interrupt increments the system time and starts time-dependent scheduling in the system. On some processors (including the MicroVAX), you cannot alter the time interval. The default is 10000.
- **Connect time** — Specify an integer in the range 1 to 3599 to indicate the seconds allowed to elapse before a circuit connection must be accepted. The default is 45.
- **Memory limit** — Specify an integer in the range 0 to 65535 to indicate the amount of physical memory in pages that is available for use by the system. The default is 0. If you specify the default, the system will use all the memory available on the target configuration.

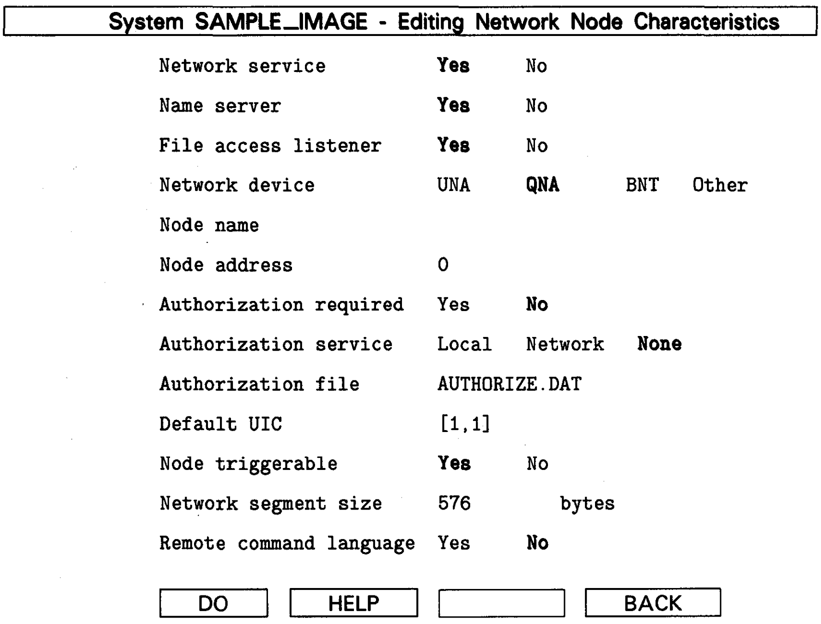
  You need to specify a limit only in special applications, such as for a system that contains a multiported memory. The multiported memory would be configured at the top of memory, and the system size you specify would not include the multiported memory. In that case, the kernel would not allocate any multiported memory for general needs. You can still use the ALLOCATE_MEMORY or ALLOCATE_SYSTEM_REGION services to access this memory.

- **EPA** — Select **Yes** to allow the collection of performance data for the system. The System Builder includes the VAXELN Performance Utility Collector. The default is **No**. For more information on the VAXELN Performance Utility, see Chapter 6.

3.4.4 Editing the Network Node Characteristics Menu

If you select the **Edit Network Node Characteristics** entry on the Main Menu, the System Builder displays the Network Node Characteristics Menu. The items on this menu define the characteristics for the Network Service and the Authorization Service. Figure 3–4 shows this menu.

Figure 3–4: Network Node Characteristics Menu



System SAMPLE_IMAGE - Editing Network Node Characteristics

Network service	**Yes**	No		
Name server	**Yes**	No		
File access listener	**Yes**	No		
Network device	UNA	**QNA**	BNT	Other
Node name				
Node address	0			
Authorization required	Yes	**No**		
Authorization service	Local	Network	**None**	
Authorization file	AUTHORIZE.DAT			
Default UIC	[1,1]			
Node triggerable	**Yes**	No		
Network segment size	576	bytes		
Remote command language	Yes	**No**		

You can edit the following network node characteristics:

- **Network service** — Select **Yes** if the System Builder is to include the Network Service in your system. The default is **Yes**. For general information on setting up a network, see the *VAXELN Run-Time Facilities Guide*.
- **Name server** — Select **Yes** if the the Network Name Service will be included in the system image. If you select **No**, the Name Service will not be included. Without the Name Service, your system cannot create or translate universal names. For example, you cannot use the NAME$UNIVERSAL argument with the EPASCAL CREATE_NAME procedure. Excluding the Name Service on stand-alone VAXELN nodes can provide savings in overhead. The default is **Yes**.
- **File access listener** — Select **Yes** if the System Builder is to include the file access listener (FAL) server in your system. The FAL allows access to files from a remote node, using the device name or a null volume name. If you select **Yes**, you should also select **Yes** for the **Network service** menu item. Otherwise, the System Builder issues an informational message. The default is **Yes**.
- **Network device** — This entry lets you select the type of driver that connects a VAXELN machine to the Ethernet in network applications. Select **UNA** for the DIGITAL Equipment UNIBUS Adapter (DEUNA). Select **QNA** for the DIGITAL Equipment Q-bus Adapter (DEQNA). Select **BNT** for the DIGITAL Equipment BI-Based Ethernet Adapter (DEBNT). The default is **QNA**.

  After you make your selection and use the DO command, the System Builder displays the Device Description Menu, letting you specify a device description for the network device driver. The menu entry **Name** displays the correct network device. If you specify **QNA** or **UNA**, the **Register address** and **Vector address** entries display the appropriate default (nonzero) values, and the **Interrupt priority** entry displays priority level 4, rather than the normal default of 5. If you select **BNT**, you must specify the correct BI bus and adapter numbers for your configuration.

  If you select **Other** for the **Network device** entry, the System Builder will include the driver for the device you specify, if you select **Yes** for the **Autoload driver** entry on the Device Description Menu. The Device Description Menu is described in Section 3.4.6.

  NOTE

  The driver you select is included in the system only if you select **Yes** for the **Network service** entry.

- **Node name** — Specify the node name by which a VAXELN node is identified in the network. The node name can have a maximum of six characters and must be unique in the network. If your system image is to be down-line loaded from your development system, you do not have to specify a node name; the target machine receives the proper node name automatically. For more information about adding node names to a network and about down-line loading, see Chapter 4.
- **Node address** — Specify the address of a VAXELN node in the network. If your system image is to be down-line loaded from your development system, you do not have to specify a node address; the target machine receives the proper node address automatically. The address can have one of three forms:

nnn	DECnet node number
aaa.nnn	DECnet area and node number
nn-nn-nn-nn-nn-nn	Ethernet address consisting of 48 bits (4 bits per hex digit)

  The *a* and *n* characters indicate digits. The default address is 0. For more information about node addresses and node numbers, see Chapter 4.
- **Authorization required** — Select **Yes** if the Network Service is to restrict inbound circuit connections to users it can authorize by communicating with the Authorization Service. Select **No** if the Network Service is not to authorize inbound connections by using the Authorization Service. The default is **No**.
- **Authorization service** — Select **Local** if an Authorization Server that serves a local node is to be included in the system. Select **Network** if an Authorization Server that serves an entire local area network is to be included in the system. More than one node can have a local server, but there should be only one network Authorization Server. The default is **None**. For general information on system security, see the *VAXELN Run-Time Facilities Guide*.
- **Authorization file** — Specify the name of the data file that the Authorization Service should use. The data file must exist on the same node as the Authorization Service or on a node that the service is authorized to access (for example, one with its own local service). The default file is AUTHORIZE.DAT.
- **Default UIC** — Specify the default user identification code (UIC) for users not explicitly authorized to use the system. The default is [1,1].

- **Node triggerable** — Select **Yes** to enable down-line load triggers. When the triggers are enabled and the Network Service is included, the EDEBUG or NCP facility can remotely trigger the system. When the Network Service is not included, an attempt to trigger will be ignored. You should select **Yes** during development so you can remotely load your system. The default is **Yes**.
- **Network segment size** — Specify an integer in the range 192 to 1470 (bytes) to indicate the maximum segment size of a message traveling over the network. This value applies to intermediate routing nodes between the source and destination of a message. The value should be the same in each VAXELN system on a particular network. This characteristic should also correspond to the EXECUTOR BUFFER SIZE on non-VAXELN systems. The default is 576.
- **Remote command language** — Select **Yes** to enable the VAXELN Command Language (ECL) to be executed from a VAX/VMS host. The System Builder includes ECL in the system. The default is **No**. For a discussion of ECL, see the VAXELN Run-Time Utilities Guide.

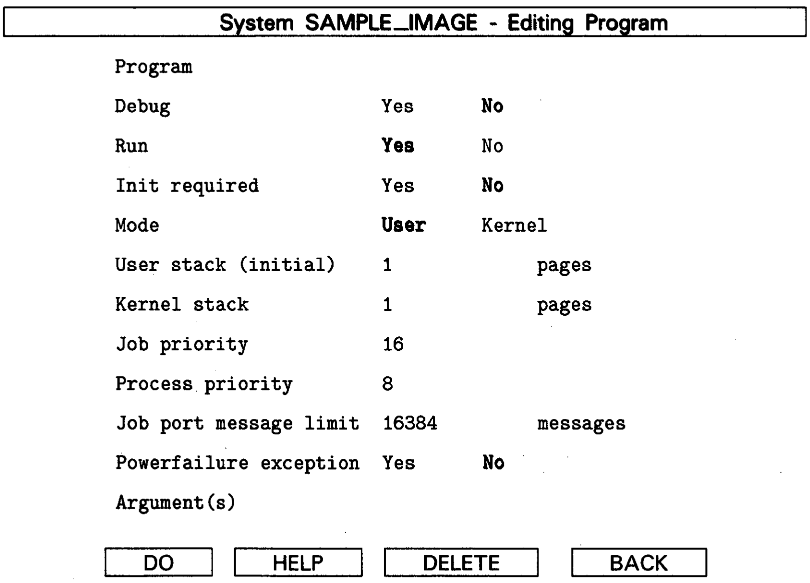
3.4.5 Editing the Program Description Menu

Each program in a system (except the kernel, the shareable run-time library images, and the autoloaded device drivers) must have a program description. If you select the **Edit Program Descriptions** entry on the Main Menu, the System Builder displays the names of programs for which descriptions exist. Each program in the list is a VAXELN job. Select a program and use the DO command. The System Builder displays the Program Description Menu for that program and lets you edit the program's description. If no descriptions exist, the System Builder displays an empty menu. Use the DO command to display the Program Description Menu and add a program description. Figure 3–5 shows this menu.

If you select the **Add Program Description** entry on the Main Menu, the System Builder displays the Program Description Menu and lets you add a program description.

The System Builder adds some program descriptions automatically. For example, if you select the **File access listener** entry on the Network Node Characteristics Menu, the System Builder automatically adds the entry's image (FALSERVER.EXE) with default program characteristics. These system images and descriptions do not appear in the list of programs on the Edit Program Descriptions Menu.

Figure 3–5: Program Description Menu



Program		
Debug	Yes [✓ No]	
Run	[✓ Yes] No	
Init required	Yes [✓ No]	
Mode	[✓ User] Kernel	
User stack (initial)	1	pages
Kernel stack	1	pages
Job priority	16	
Process priority	8	
Job port message limit	16384	messages
Powerfailure exception	Yes [✓ No]	
Argument(s)		

The default menu settings are good starting points for most values; try using the defaults before you change them.

Descriptions of the Program Description Menu entries follow:

- **Program** — Specify the name of the program image you are describing. For example, if the image file is named MYDRIVER.EXE, specify MYDRIVER. If you want the system to associate different characteristics with different jobs running the program, you can describe the program more than once. For example, you might want to include two descriptions of a program so you can debug one version but not the other.

  To include multiple descriptions, use the same image name in each description. The System Builder loads the image once. The different descriptions, however, apply to the different jobs created to run the image.

The System Builder distinguishes descriptions in the system map. The system map indicates any additional descriptions of a program by the program name followed by a semicolon and a number. The number 1 indicates the first additional description; for the original description, the program name does not have a suffix. You might need to specify a name with a suffix in calls to the CREATE_JOB procedure or with the debugger's CREATE JOB command.

- **Debug** — Select **Yes** if a job that runs the program is to pass control to the debugger when the job is first eligible to run. The default is **No**. See Chapter 5 for details about debugging.
- **Run** — Select **Yes** if a job running the program is to be created and eligible to start when the system starts executing. The default is **Yes**.
- **Init required** — Select **Yes** if the program is to execute initialization code — that is, if the program calls the INITIALIZATION_DONE procedure. If several programs call this procedure, the System Builder places them in order of job priority. (The System Builder, however, gives debuggers and device drivers predetermined priorities and places them accordingly.) The default is **No**.
- **Mode** — Select **Kernel** if the program is a device driver (programs calling CREATE_DEVICE) or if the program uses routines that require kernel mode, for example, the ALLOCATE_MAP, MFPR, MTPR, and FREE_MAP routines. Select **User** mode (the default) if the program does not require kernel mode.
- **User stack (initial)** — Specify an integer in the range 1 to 32767 to indicate an initial stack size for user-mode calls to user-defined procedures and to most predeclared procedures. The default is 1. The kernel extends the stack as needed while a job executes. User mode programs run off the user stack except when they call kernel procedures or change to kernel mode.
- **Kernel stack** — Specify an integer in the range 1 to 32767 to indicate the stack size for programs that run in user mode and call kernel procedures or that run in kernel mode. The default is 1. However, most kernel-mode programs require more than one page. The kernel does not extend the kernel stack automatically. If a program attempts to use too much of the kernel stack, the process receives an exception. You can avoid the exception by using the ALLOCATE_STACK kernel service.
- **Job priority** — Specify an integer in the range 0 to 31 to indicate the program's job priority. The highest priority is 0, the lowest priority is 31, and the default is 16. Try running the program with the default priority. If necessary, change the priority to optimize the system's performance.

NOTE

Certain system jobs and drivers run at low priority. For example, the remote debugger runs at priority 3 and the Ethernet driver at priority 1. Assigning a job a higher priority than these system jobs and drivers may affect the system adversely. For example, the remote debugger will not get control.

- **Process priority** — Specify an integer in the range 0 to 15 to indicate the program's process priority. The highest priority is 0, the lowest priority is 15, and the default is 8. The process priority is the initial priority of the master process and the subprocesses it creates. Try running the program with the default priority. If necessary, change the priority to optimize the system's performance.
- **Job port message limit** — Specify an integer in the range 0 to 16384 to indicate the maximum number of messages that can be queued to the job's port without generating a wait or an error. The default is 16384.
- **Powerfailure exception** — Select **Yes** if the program is to receive the exception KER$_POWER_SIGNAL when the processor restarts after a power failure. Assign this exception to programs that will establish an exception handler for KER$_POWER_SIGNAL. This type of handler lets you take general, systemwide action when power fails; for example, it lets you reset the system time. Device drivers generally need to handle power recovery with interrupt service routines. This use of interrupt service routines is not affected by your selection of this program description. The default is **No**.
- **Argument(s)** — Specify the program's arguments. Program arguments are strings. If an argument includes embedded spaces, enclose the argument in quotation marks (for example, *"my arg"*). Separate multiple arguments with commas. The requirements for and the meaning of program arguments depend on the program. For example, device driver programs generally require arguments that supply names for the devices they are to control.

  The most frequently used argument is a file specification used to redirect terminal input or output to a file or a device other than the controlling terminal. The first three program arguments provide equivalence strings for the reserved file specifications SYS$INPUT:, SYS$OUTPUT:, and SYS$ERROR:. These file specifications redirect terminal input, terminal output, and error message output, respectively. You can omit the equivalence string for one of these file specifications by specifying a null program argument in that argument's position. If you specify a null program argument, the argument defaults to CONSOLE:. You can also use this method of redirecting I/O with the CREATE_JOB procedure.

In EPASCAL, the program arguments associated with the predeclared file variables INPUT and OUTPUT are associated with SYS$INPUT: and SYS$OUTPUT:, respectively. The order in which you specify INPUT and OUTPUT does not matter. There is no default for SYS$ERROR: in EPASCAL. (See the *VAXELN Pascal Language Reference Manual*) for more information about I/O redirection in EPASCAL.

NOTE

> Redirecting terminal I/O in a mixed-language application, in which terminal I/O is performed from more than one language, is unsupported.

If you enter more arguments than fit on the screen, the argument display scrolls to the left. Use the Left Arrow (←) and Right Arrow (→) keys to scroll the argument list. Use control commands to move around the argument list, delete arguments, or clear the screen (see Section 3.4).

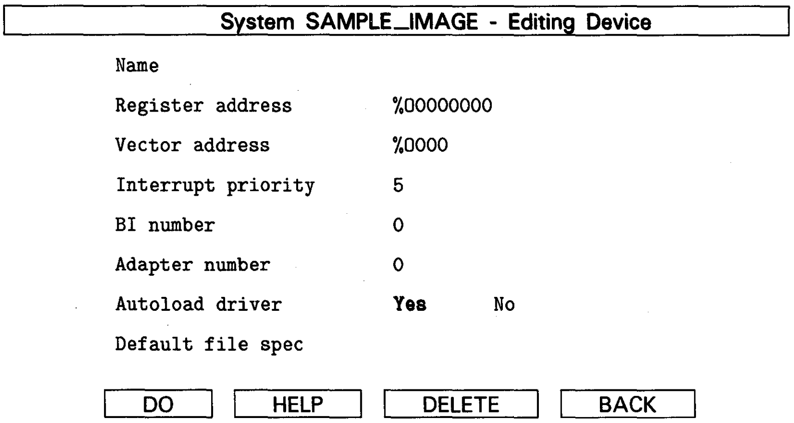
3.4.6 Editing the Device Description Menu

Each device that is part of the target machine's hardware configuration must have a device description. Device descriptions consist of a device name, register address, vector address, interrupt (bus-request) priority, BI node number, and adapter number. In most cases, device descriptions cause the associated device driver to be built into the system.

If you select the **Edit Device Descriptions** entry on the Main Menu, the System Builder displays the names of devices for which descriptions exist. Select a device and activate the DO command. The System Builder displays the Device Description Menu for that program and lets you edit the device's description.

If no descriptions exist, the System Builder displays an empty menu. Activate the DO command to display the Device Description Menu and add a device description. Figure 3–6 shows the menu.

Figure 3–6: Device Description Menu



```
                 System SAMPLE_IMAGE - Editing Device

        Name
        Register address        %00000000
        Vector address          %0000
        Interrupt priority      5
        BI number               0
        Adapter number          0
        Autoload driver         Yes      No
        Default file spec

        [ DO ]  [ HELP ]  [ DELETE ]  [ BACK ]
```

If you select the **Add Device Description** entry on the Main Menu, the System Builder displays the Device Description Menu and lets you add a device description.

You must specify the control/status register addresses, interrupt vector addresses, priorities, BI number, and adapter number for bus devices as described in the appropriate device's hardware manual for your system configuration. For devices supported by DIGITAL-supplied drivers, see Table 3–4. (The device names listed in Table 3–4 are the conventional names for the first device controller of the specified type and cause the appropriate driver to be loaded.)

Table 3–4: Device Information

Device	Description	Name	Register Address	Vector Address	Priority	BI number	Adapter number
ADV11	Analog input/output board.[1]	ADV	%O170400	%O400	4	0	0
AXV11	Analog input/output board.[1]	AXV	%O170400	%O400	4	0	0

[1]Do not autoload driver.

Table 3–4 (Cont.): Device Information

Device	Description	Name	Register Address	Vector Address	Priority	BI number	Adapter number
CXY08	Q-bus-based quad-height 8-line asynchronous full-duplex serial multiplexer with modem support.	TTA	%O0760440	%O300	4	0	0
CXA16	Q-bus-based 16-line asynchronous full-duplex serial multiplexer. Modems are not supported.	TTA	%O0760440	%O300	4	0	0
CXB16	Q-bus-based 16-line asynchronous full-duplex serial multiplexer. Modems are not supported.	TTA	%O0760440	%O300	4	0	0
DEBNA	BI-based Ethernet controller.	XBA	%O000000	%O000	4	0–3	BI node number
DEQNA	DIGITAL Q-bus-to-Ethernet Adapter.	XQA	%O774440	%O120	4	0	0
DEUNA	DIGITAL UNIBUS-to-Ethernet Adapter.	XEA	%O774510	%O120	5	See [2]	See [2]
DHQ11	MicroVAX multiplexer. Modems are supported on all eight lines.	TTA	%O760440	%O300	4	0	0
DHV11	MicroVAX multiplexer. Modems are supported on all eight lines.	TTA	%O760440	%O300	4	0	0

[2]For devices connected to the BI–to–UNIBUS Adapter (DWBUA), specify the BI and adapter numbers listed for the DWBUA.

Table 3–4 (Cont.): Device Information

Device	Description	Name	Register Address	Vector Address	Priority	BI number	Adapter number
DLVJ1	Asynchronous serial interface — same as the DLV11-J. [1]	DLV	%O176500	%O300	4	0	0
DMF-32	VAX-11/730 or VAX-11/750 line printer, terminals, DR11C parallel I/O. LC is the conventional name for the printer controller, and TT is for terminals.	LCA TTA	%O760340[3]	%O320[3]	5	0	0
DRB32	BI-based DMA parallel interface.[1]	DRB	%O000000	%O000	4	0–3	BI node number
DRQ3B	Q-bus-based DMA parallel I/O interface.[1]	DRQ	%O760740	%O300	4	0	0
DRV11	High-density parallel interface.[1]	DRV	%O764160	%O340	4	0	0
DWBUA	BI-to-UNIBUS adapter.	BUA	%O000000	%O000	4	0–3	BI node number
DZQ11	MicroVAX multiplexer. 4 asynchronous lines with modem control.	TTA	%O760100[4]	%O300	4	0	0
DZV11	MicroVAX multiplexer. 4 asynchronous lines with modem control.	TTA	%O760100[4]	%O300	4	0	0
IEQ11	Dual IEC/IEEE 488 bus controller.	GPA	%O764100	%O270	4	0	0

[1]Do not autoload driver.

[3]Specify the same addresses for every device on the same DMF–32.

[4]The DZV11 is shipped with a register address of %O760010, which must be changed to %O760100 for a MicroVAX.

Table 3–4 (Cont.): Device Information

Device	Description	Name	Register Address	Vector Address	Priority	BI number	Adapter number
KDA50	MicroVAX disk controller. The KDA50 controller supports RA*xx*-series disks.	DUA	%O772150	%O154	4	0	0
KDB50	BI RA*xx*-series disk adapter.	BDA	%O000000	%O000	4	0–3	BI node number
KWV11	Programmable real-time clock.[1]	KWV	%O170420	%O440	4	0	0
LPV11	MicroVAX line printer controller.	LPA	%O777514	%O200	4	0	0
RQC25	Q-bus-based disk controller for the RC25.	DUA	%O772150	%O154	4	0	0
RQDX*n*	MicroVAX disk controller. The RQDX*n* controller supports both RX*nn* diskettes and RD*nn* Winchester disks.	DUA	%O772150	%O154	4	0	0
RUC25	Uni-bus-based disk controller for the RC25.	DUA	%O772150	%O154	4	0	0
TQK50	Cartridge tape controller for the TK50.	MUA	%O774500	%O260	4	0	0
TQK70	Cartridge tape controller for the TK70.	MUA	%O774500	%O260	4	0	0
TU58	Console tape for VAX–11/730 and VAX–11/750 processors.	DDA	%O000000[5]	%O360	4	0	0

[1]Do not autoload driver.

[5]Do not specify a register address, because the device is controlled by internal processor registers.

Table 3–4 (Cont.): Device Information

Device	Description	Name	Register Address	Vector Address	Priority	BI number	Adapter number
TU81	Reel tape system.	MUA	%O774500	%O260	5	0	0
UDA50	UNIBUS disk adapter for RA*xx*-series disks on the VAX–11/730 or VAX–11/750.	DUA	%O772150	%O154	5	0	0
VAX–11/730 integrated controller	A controller that supports the RB02/RB80 disk, which is the same as the RL02/R80 disks.	DQA	%O775606	%O250	5	0	0

In cases where more than one device controller is permitted on the same backplane (such as the DZV11 multiplexer in MicroVAX systems), the addresses shown in Table 3–4 are for the first such controller. For further information about additional controllers, see the appropriate device's hardware manual or, for MicroVAX devices, the *MicroVAX I Owner's Manual* or *MicroVAX II Owner's Manual.*

Devices with multiple interrupt vectors require one device description; the additional vectors are specified with arguments to the CREATE_DEVICE procedure.

Descriptions of the Device Description Menu entries follow:

- **Name** — Specify a device name that describes the device controller. Programs use the device name as an argument to the procedure CREATE_DEVICE (usually to call a driver program). For example, if the name of a terminal controller is TTA, its driver program is given TTA as a program argument. The driver program uses the device name as an argument to the CREATE_DEVICE procedure. When specifying terminals, you must describe individual lines by using terminal descriptions, and you must name individual lines with a controller name and a line number (for example, TTA1). See Section 3.4.7 for information about terminal characteristics.

In the VAXELN documentation, conventional device names are used. For example, DQA1 is used for a disk controlled by the VAX-11/730 Integrated Disk Controller. However, you can specify your own device name. (See also the discussions below of the **Autoload driver** and **Default file spec** menu items for the effect the device name has on automatic device driver loading.)

You must use the device name you specify consistently in various contexts (that is, in different types of calls).

- **Register address** — Specify a register address in the range %O0000000 (the default) to %O0777777 to indicate the physical address of the device's first device control register. Use 18-bit addresses for Q-bus and UNIBUS devices, including devices connected to a DIGITAL BI-to-UNIBUS Adapter. You can specify the address in decimal, octal (%O), or hexadecimal (%X). For the correct value, see Table 3–4 or consult the device's hardware manual.

  For BI processors, the kernel assigns the proper number. Therefore, you must use the default value, %O0000000.

  NOTE

  The register address is extended, letting you specify complete VAX physical addresses in the I/O space for synchronous backplane interconnect (SBI) interface boards.

  A driver program can obtain the address by using an output argument with the CREATE_DEVICE procedure.

- **Vector address** — Specify a vector address in the range %O000 (the default) to %O776 to indicate the address of the device's first interrupt vector. For UNIBUS and Q22-bus devices, specify the vector that the device asserts on the bus when its interrupt request is acknowledged. The VAX processor uses the address as an index to the correct page of the system control block (SCB). The System Builder fills in the SCB vector in a call to the CREATE_DEVICE procedure. For the correct value, see Table 3–4 or consult the device's hardware manual.

  For BI processors, the kernel assigns the proper number. Therefore, you must use the default, value, %O000.

  A driver program can obtain the address of the vector of the device, or the addresses for multiple-vector devices, by using an output argument with the CREATE_DEVICE procedure.

- **Interrupt priority** — Specify an integer in the range 4 to 7 to indicate the device's bus-request priority. The highest priority is 4, the lowest priority is 7, and the default priority is 5. These values correspond to the VAX interrupt priority levels 14 (hexadecimal) to 17 (hexadecimal). For the correct value, see Table 3–4 or consult the device's hardware manual.

  You can obtain the resulting interrupt priority level by specifying an output argument with the CREATE_DEVICE command.

- **BI number** — The BI number indicates the number of the BI bus on which the device is located. If you are editing a description for an 8800 or 8700 processor, specify an integer in the range 0 to 3. If you are editing a description for an 8550 or 8500 processor, specify 2 or 3. You must also use the default register and vector addresses (%O0000000 and %O000), an adapter number, and possibly the default file specification for the autoload driver.

  If you are editing a description for a UNIBUS device connected to a DIGITAL BI-to-UNIBUS Adapter, specify the BI number containing the UNIBUS adapter. In addition, you must specify an 18-bit register address, a UNIBUS vector for the vector address, and the BI node number for the UNIBUS adapter as the adapter number.

  In all other cases, specify 0 (the default).

  NOTE

  If you specify node 0 and the register address %O000000, the console is assumed.

- **Adapter number** — If you are editing a description for an 8800, 8700, 8550, or 8500 processor, specify an integer in the range 0 to 15. You must also specify the default register and vector addresses (%O0000000 and %O000), a BI number, and the appropriate default file specification for the autoload driver.

  If you are editing a description for a UNIBUS device connected to a DIGITAL BI-to-UNIBUS Adapter, specify the BI node number for the UNIBUS adapter. In most cases, this is 0. In addition, you must specify an 18-bit register address, a UNIBUS vector for the vector address, and a BI number containing the UNIBUS adapter.

  If your target processor is a dual UNIBUS 11/750, specify 0 or 1 for the UNIBUS adapter number.

  In all other cases, specify 0 (the default) for the adapter number.

- **Autoload driver** — Select **Yes** if the System Builder is to include the appropriate device driver image in your system image. Select **No** if you want to include the driver in the system by entering it on the Program Description Menu. You might want to disable the Autoload driver feature while you are developing a new device driver so you can assign the Debug characteristic to the program.

  If you select **Yes**, the System Builder searches the list of program descriptions for a program that has the **Kernel** and **Init required** characteristics and that has at least one program argument matching the device name. The System Builder gets a driver (if one exists) from ELN$:*cc*DRIVER.EXE, where *cc* is the first two characters in the device name.

  NOTE

  You can override the default by specifying a **Default file spec**.

  The name specified by the device description is passed to the driver as a program argument. A program description is provided that has appropriate program characteristics, including:

  - A high job priority (such as 4)
  - A 4-page kernel stack
  - The Kernel characteristic
  - The Init required characteristic

  Use the /MAP qualifier to examine the results. The Autoload driver feature is the easiest way to include drivers, including user-written drivers.

  The default is **Yes**.

  The selection and loading of terminal drivers other than the console driver is controlled by the **Terminal type** entry on the Terminal Description Menu. However, you must supply a device description for the terminal controller, such as the DMF–32 asynchronous controller. You can then use the controller's device name with an appended digit as a **Terminal name** on the Terminal Description Menu to designate the characteristics of the particular terminal attached to that line.

  NOTE

  When autoloading a terminal driver, you must specify at least one terminal name on the Add Terminal Description Menu for each terminal controller.

You might want to load a line printer driver with an explicit program description, which lets you specify a universal name as a second program argument. Assigning a universal name lets you access the printer from remote nodes.

- **Default file spec** — By default, the system derives the name of the device driver file from the first two characters of the device name (ELN$:*cc*DRIVER.EXE). If you select **Yes** for the **Autoload driver** entry, you can use a different device, directory, or file name. For example, separate versions of the DU or MU drivers are required by some machine architectures. The **Default file spec** entry lets you use one driver device name for different machine types.

  When entering a **Default file spec**, you must specify at least a file name. The device, directory, and file type are optional; the default device/directory is the logical name ELN$, and the default file type is EXE.

  Specify BDDRIVER for a KDB50. The System Builder uses the default for all other drivers.

3.4.7 Editing the Terminal Description Menu

Each terminal connected to an asynchronous serial controller line must have a terminal description. You must include a device description for the asynchronous controller (for example, the DMF–32 or DZV11).

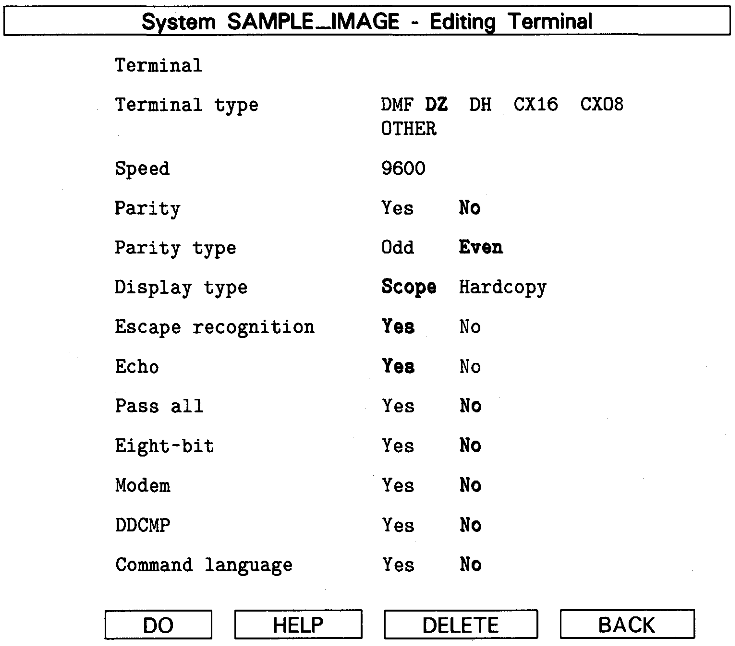
NOTE

Describe the console terminal on the Console Characteristics Menu.

If you select the **Edit Terminal Descriptions** entry on the Main Menu, the System Builder displays the names of terminals for which descriptions exist. Select a terminal and activate the DO command. The System Builder displays the Terminal Description Menu for that terminal and lets you edit the terminal's description.

If no descriptions exist, the System Builder displays an empty menu. Activate the DO command to display the Terminal Description Menu and add a terminal description. Figure 3–7 shows the menu.

Figure 3–7: Terminal Description Menu



System SAMPLE_IMAGE - Editing Terminal

Terminal	
Terminal type	DMF **DZ** DH CX16 CX08 OTHER
Speed	9600
Parity	Yes **No**
Parity type	Odd **Even**
Display type	**Scope** Hardcopy
Escape recognition	**Yes** No
Echo	**Yes** No
Pass all	Yes **No**
Eight-bit	Yes **No**
Modem	Yes **No**
DDCMP	Yes **No**
Command language	Yes **No**

If you select the **Add Terminal Description** entry on the Main Menu, the System Builder displays the Terminal Description Menu and lets you add a terminal description.

You can edit the following entries on the Terminal Description Menu:

- **Terminal** — Specify the name of a terminal, using a controller device name followed by a unit number; for example, TTA0 is the first line on controller TTA.
- **Terminal type** — This entry specifies the type of controller to be used for terminals. Table 3–5 describes the choices displayed.

Table 3–5: Terminal Types

Entry	Description
DMF	The asynchronous lines on a UNIBUS DMF–32 controller
DZ	The DZV11 interface on a MicroVAX processor
DH	The DHV11 interface on a MicroVAX processor
CX16	The CXA16 and CXB16 interfaces on the Q-bus
CX08	The CXY08 interface on the Industrial VAX

Your selection indicates the terminal driver for the controller type that is to be loaded. The terminal name (for instance, TTA1) designates a terminal in programs. The default is **DZ**.

If you select **OTHER**, the System Builder searches for a device name whose first three characters match the first three characters of the name you specify for the **Terminal** entry. This option lets you create descriptions for user-written terminals.

- **Speed** — Specify the baud rate for input and output on the terminal line. Possible values are:

50	134	600	2000	4800	19200
75	150	1200	2400	7200	38400
110	300	1800	3600	9600	

The default is 9600.

- **Parity** — Select **Yes** to enable parity checking on the terminal line. The default is **No**.
- **Parity type** — If parity checking is enabled, select **Odd** if you want odd parity. Select **Even** if you want even parity. The default is **Even**.
- **Display type** — Select **Scope** if the attached terminal is a cathode ray tube (CRT) terminal, such as a VT100 or VT200. Select **Hardcopy** if the attached terminal is a hard-copy terminal. The default is **Scope**. This setting is ignored for a DDCMP line.
- **Escape recognition** — Select **Yes** if the terminal driver program is to check whether the format of escape sequences conforms to ANSI format. This setting is ignored on a DDCMP line. The default is **Yes**.
- **Echo** — Select **Yes** if characters are to be echoed on the terminal for read operations. This setting is ignored for a DDCMP line. The default is **Yes**.

- **Pass all** — Select **Yes** if control characters are to be passed to user programs as ordinary input instead of being interpreted by the driver program. This setting is ignored for a DDCMP line. The default is **No**.
- **Eight-bit** — Select **Yes** if the attached terminal uses 8-bit ASCII characters. This setting is ignored for a DDCMP line. The default is **No**.
- **Modem** — Select **Yes** if a modem is attached to the line. Modems are supported only on the DHV11 and DMF–32 controllers. The default is **No**.
- **DDCMP** — This entry specifies whether the terminal line should use the DIGITAL Data Communications Message Protocol (DDCMP) for asynchronous DECnet communication with another system. Select **Yes** if the line acts as a point-to-point full-duplex DDCMP link. Select **No** if the line acts as a regular terminal line. The default is **No**.
- **Command language** — Select **Yes** to enable the entering of ECL commands at the terminal. The System Builder includes ECL in the system. The default is **No**.

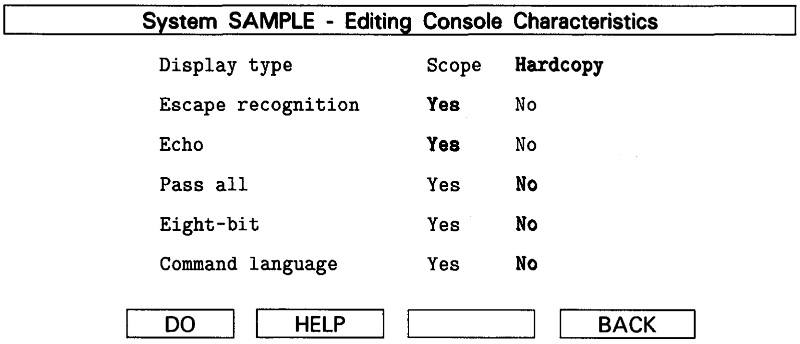
3.4.8 Editing the Console Characteristics Menu

If you select the **Edit Console Characteristics** entry on the Main Menu, the System Builder displays the Console Characteristics Menu. Items on this menu have the same meanings as they do for terminal descriptions, but the settings apply only to the console terminal on the target machine. Figure 3–8 shows this menu.

The entries you can describe on the Console Characteristics Menu follow:

- **Display type** — Select **Scope** if the attached terminal is a CRT terminal. Select **Hardcopy** if the attached terminal is a hard-copy terminal. The default is **Hardcopy**.
- **Escape recognition** — Select **Yes** if the terminal driver program is to check that the format of escape sequences for input conforms to ANSI format. The default is **Yes**.
- **Echo** — Select **Yes** if input characters are to be echoed on the terminal for read operations. The default is **Yes**.

Figure 3–8: Console Characteristics Menu



System SAMPLE - Editing Console Characteristics

Display type Scope [✓ Hardcopy]
Escape recognition [✓ Yes] No
Echo [✓ Yes] No
Pass all Yes [✓ No]
Eight-bit Yes [✓ No]
Command language Yes [✓ No]

- **Pass all** — Select **Yes** if control characters are to be passed to user programs as ordinary input instead of being interpreted by the driver program. The default is **No**.
- **Eight-bit** — Select **Yes** if the attached terminal uses 8-bit ASCII characters. The default is **No**.
- **Command language** — Select **Yes** to enable the entering of ECL commands at the console terminal. The System Builder includes ECL in the system. The default is **No**.

3.4.9 Editing the Error Log Characteristics Menu

If you select the **Edit Error Log Characteristics** entry on the Main Menu, the System Builder displays the Error Log Characteristics Menu. This menu enables error and event logging. You use the VAX/VMS Error Log Utility to process the information generated by your selections on this menu. The reports produced by the VAX/VMS Error Log Utility are primarily intended to assist DIGITAL field service personnel.

VAXELN error log files have entries for the following events:

- Errors: device errors, device timeouts, machine checks, bus errors, memory errors, asynchronous write errors, undefined interrupts, and bugchecks
- Volume changes: volume mounts and dismounts

- System events: cold start-ups, warm start-ups, system failures, and time stamps

The error log file can be created on the target system or on a remote system over the network.

To create the error log file on a remote node, you must use the Error Log Server (ELSE) (see Appendix B). When you create the error log file on the target, you must transfer the file to a VAX/VMS system in order to use the VAX/VMS Error Log Utility. If the target is connected to a VAX/VMS system by the Ethernet, you can use the DCL COPY command to transfer the file. Alternatively, you can include ECL in your VAXELN system and use the ECL COPY command to write the file to a storage medium that can be moved to a VAX/VMS system.

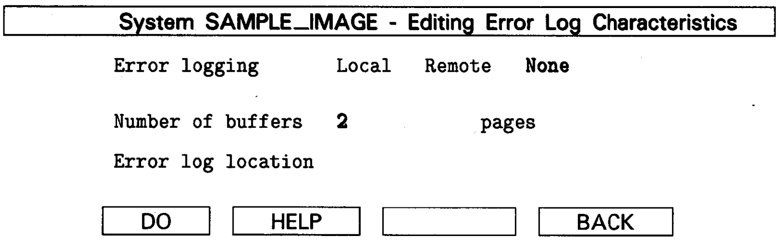
Locally created error log files have a time format that differs from remotely created files. VAXELN only records the elapsed time from the system boot. Files created on VAX/VMS systems have absolute times.

For descriptions of the VAX/VMS Error Log Utility, see the following:

- *VAX/VMS Error Log Utility Reference Manual*
- *VAX/VMS System Manager's Reference Manual*
- *Guide to VAX/VMS System Management and Daily Operations*

Figure 3–9 shows the Error Log Characteristics Menu.

Figure 3–9: Error Log Characteristics Menu



System SAMPLE_IMAGE - Editing Error Log Characteristics

```
Error logging       Local   Remote   None

Number of buffers   2           pages

Error log location
```

You can edit the following entries on the Error Log Characteristics Menu:

- **Error logging** — If you select **Local**, the System Builder includes local error logging in the system. If you select **Remote**, the System Builder includes remote error logging in the system. **None** is the default.

  NOTE

  With the possible exception of applications with very critical real time requirements, error logging will have minimal impact on run-time performance.

- **Number of buffers** — Specify an integer in the range 2 to 65535 to indicate the number of 512-byte pages allocated for error log buffers. The **Number of buffers** is ignored if you deselect error logging. The default and minimum is 2.
- **Error log location** — Specify the location where the error log file is to be written. If you select **Remote**, the location is the node number of the remote system. If you select **Local**, the location is a file specification. The device name must be entered in the **Disk/volume names** field of the System Characteristics Menu.

  To assist field service personnel, DIGITAL recommends that the file specification have the following form:

  [sys*n*.syserr]errorlog.sys

  where *n* is a hexadecimal digit 0 through F. An example of a complete file specification for a local error log file would be:

  DUA0:[SYS2.SYSERR]ERRORLOG.SYS

You may want to build two versions of a system, one with and one without error logging. Then if problems occur, you can run the version with error logging to analyze the problem. If a VAXELN system does not include error logging and a fatal event occurs, bugcheck information will be displayed at the system console, if one is attached.

Chapter 4

Booting and Down-Line Loading

After you use the System Builder to create a system image, you can boot the image from a disk or down-line load the image from your host system. This chapter explains how to perform these operations.

The VAXELN installation procedure creates the system images ICP.SYS for booting and ICP_DOWNLINE.SYS for down-line loading (see the *VAXELN Installation Manual*). You should boot or down-line load one of these images once to ensure that the installation was successful.

If you prefer to boot a system image (ICP) from a disk, you can copy the image to a target disk, carry the medium to your target machine, and boot the image by following instructions provided in Section 4.1.

If your host and target machines are connected by the Ethernet, you can down-line load your system image (ICP_DOWNLINE) as described in Section 4.2.

4.1 Booting Systems from Disks

This section explains how to boot a system image from a disk. The examples illustrate the procedure for booting the system image ICP.SYS. However, you can use the same procedure to boot system images that you create.

To boot a system image:

1. Use the COPYSYS command procedure in the ELN$ directory on your host machine to copy the system image to a target device (tape cartridge, disk, or diskette).

2. Transfer the storage medium to the target machine.
3. Boot the volume.

NOTE

> To use the COPYSYS command procedure, you must set the EBUILD system characteristic **Boot method** to **Disk** (for disks, diskettes, or tape cartridges). See Section 3.4.3 for information about system characteristics.

The COPYSYS.COM command procedure resides in the directory ELN$. Use this command procedure to initialize and load a target device, such as a TU58 cartridge. To invoke the command procedure, type:

```
$ @ELN$:COPYSYS
```

The procedure prompts you for the name of a system image file and an output disk. To prepare a bootable copy of the system image ICP.SYS on a TU58 cartridge, respond to the prompts as follows:

```
System image file: ELN$:ICP
Output disk: CSA1
```

The command procedure then asks if you want to initialize the disk. If you are copying to an unused disk or cartridge, you must type Y. If you are reusing a disk or cartridge, you can type N (the default).

```
Initialize the disk? (Y/N) [N]: Y
$
```

If you receive the error message "No such device" and you are using the console device, the console device might not be connected. Your system manager can connect the console by entering the command:

```
$ MCR SYSGEN CONNECT CONSOLE
```

If you do not need to initialize your target device, you can enter your input on one line. For example:

```
$ @ELN$:COPYSYS ELN$:ICP CSA1
```

After you copy the image, transfer the TU58 cartridge to the target machine and boot the system image by specifying the console boot command (B) and the name of the device you are booting. For example, to boot an image on a TU58 cartridge on an 11/750 target machine, specify:

```
>>> B DD0
```

NOTE

> To access an 11/750 TU58 device from a VAXELN application, the application must use DDA1 for the device name.

After you enter the boot command, ICP.SYS begins running, and the system displays the following:

```
%VAXELN system initializing

       VAXELN  V3.0

Testing Pascal run time routines...
All Pascal run time routines correct
Testing C run time routines...
All C run time routines correct
Testing time manipulation routines...
All time manipulation routines correct
Testing system memory services...
All system memory services correct
Testing exception handling...
Exception handling correct
Testing process synchronizing and scheduling...
Process synchronizing and scheduling correct
Testing message passing and job scheduling...
Message passing and job scheduling correct

SUCCESSFUL COMPLETION OF VAXELN TESTING
```

Booting the system image ICP.SYS takes approximately five minutes.

Table 4–1 lists the target devices you can boot and their corresponding names.

Table 4–1: Target Device Names

Device	Name
RAnn disk	DU*n*
RC25 disk	DU*n*
RDnn disk	DUA*n*
RL02 disk	DQ1
RX33 diskette	DUA*n*
RX50 diskette	DUA*n*
TK50 cartridge	DD0

Table 4–1 (Cont.): Target Device Names

Device	Name
TK70 cartridge	DD0
TU58 cartridge	DD0
TU81 reel to reel	DD0

4.2 Down-Line Loading

If your host and target machines are connected by the Ethernet, you can use the Ethernet to load a system image onto the target machine and boot the system image. Using the Ethernet to load a system image is called *down-line loading*. When you down-line load a VAXELN system, you use a down-line load bootstrap loader on the target machine and the DECnet network facilities on the host development system. These two software components use network communication hardware to copy a VAXELN system image file from the host development system to the main memory of the target machine. Once the VAXELN system is in the target machine's memory, the target machine gets control of the processor and begins execution.

The VAXELN system need not contain the Network Service to be down-line loaded. However, you must include the Network Service to enable network communication between the VAXELN system and other systems on the same network. For more information about the Network Service, see the *VAXELN Run-Time Facilities Guide*.

The rest of this chapter describes the preliminary steps and the procedure for down-line loading system images onto target systems, using the Ethernet. The examples illustrate the procedure for down-line loading the system image ICP_DOWNLINE.SYS. However, you can use the same procedure to boot system images that you create.

4.2.1 Preliminary Steps

Before you down-line load a system image, you must set up your host and target machines by completing the following steps:

1. Install communication hardware on the host and target machines.
2. Configure and install DECnet-VAX software on the host system.
3. Test communication between the host and target machines.
4. Add the target machine's description to the host system's network node data base.
5. Configure and install the down-line load bootstrap loader on the target machine.

Before continuing, you should become familiar with the network control program (NCP). This utility is the principal tool used to control the network software and hardware and is fully described in the *DECnet-VAX System Manager's Guide*. The following sections explain how to:

- Install communication hardware on a target machine
- Configure a host for down-line loading
- Add a target machine to the host's network node data base
- Configure the bootstrap loader

4.2.1.1 Installing Communication Hardware on the Target Machine

Install communication hardware at the default I/O bus address on the target processor. Table 4–2 lists the address assumed by the down-line load bootstrap loader for each particular hardware device.

Table 4–2: Datalink Device Default Addresses

Device	Address (Octal)
DEBNT	None
DEQNA	774440
DEUNA	774510

4.2.1.2 Configuring a Host for Down-Line Loading

To configure your VAX/VMS host for down-line loading, issue the following commands:

```
$ RUN SYS$SYSTEM:NCP
NCP> DEFINE LINE UNA-0 SERVICE ENABLED
NCP> DEFINE CIRCUIT UNA-0 SERVICE ENABLED
NCP> SET LINE UNA-0 STATE OFF
NCP> SET LINE UNA-0 ALL
NCP> SET CIRCUIT UNA-0 STATE OFF
NCP> SET CIRCUIT UNA-0 ALL
```

These commands enable the host to recognize boot-request messages from the target system.

NOTE

> If you are configuring a MicroVMS system, use QNA–0 instead of UNA–0 for the service line and the service circuit. If you are configuring an 8800, 8700, 8550, 8530, or 8500 system with a DEBNT controller, use BNA–0.

4.2.1.3 Adding the Target Machine to the Host's Network Node Data Base

You must describe the target machine in the host system's network node data base. Use the NCP utility to store the target machine's node address, node name, Ethernet hardware address, and host load device name. Invoke the utility as follows:

```
$ RUN SYS$SYSTEM:NCP
NCP>
```

Use the DEFINE command to add target data to the permanent data base. The following example uses FRED as the target node name:

```
NCP> DEFINE NODE FRED ADDRESS 5 SERVICE CIRCUIT UNA-0
NCP> DEFINE NODE FRED HARDWARE ADDRESS AA-00-03-00-00-E1
```

NOTE

> To use the DEFINE command, you must specify a system user identification code (UIC) or set the system privilege option (SYSPRV).

If you are configuring a MicroVMS system, use QNA-0 instead of UNA-0 for the service circuit. If you are configuring an 8800, 8700, 8550, 8530, or 8500 system with a DEBNT contrtoller, use BNA-0.

Each node in your network must have a unique address and name. The service circuit is the name of the host system's hardware device controller, connecting the host system to the target machine.

The hardware address is required for down-line loading using the Ethernet and is the Ethernet address contained in ROM on the target machine's Ethernet hardware controller. You can usually locate this address on the controller board, but if you cannot, contact your field service representative, who can provide the address by running the controller's diagnostic package.

Once the target machine has been added to the host system's permanent data base, use the SET command to copy the information to the current data base. For example:

```
NCP> SET NODE FRED ALL
```

NOTE

To use the SET command, you must first set the operator privilege option (OPER).

The target machine's description remains in both data bases, even after the host system is rebooted.

4.2.1.4 Configuring and Installing the Bootstrap Loader

You must configure and install the down-line load bootstrap loader on the target machine. VAX-11/730- and VAX-11/750-family processors use the console storage medium (TU58) to store the bootstrap loaders. MicroVAX processors store the down-line loader in the boot ROM. VAX 8800, 8700, 8550, 8530, and 8500 processors store the loader on a P/OS diskette.

To install the down-line load bootstrap loader on a TU58 console tape, use the VAXELN NEWBOOT command procedure. You should run this procedure from the system manager's account because some operations require the CMKRNL privilege.

The NEWBOOT procedure copies the bootstrap loader's image file to the console medium. This command procedure prompts you for the bootstrap load device (XE=DEUNA), the device containing the console medium on which the loader is to be installed, and the processor type of the target machine. For example:

```
$ SET DEFAULT ELN$
$ @NEWBOOT
Bootstrap device [XE]:
Console media device [CSA1]:
Processor type [730]:
Set default bootstrap? (Y/N) [Y]:
```

After the command procedure copies the loader files to the console medium, the loader installation is complete.

No configuration is required for a MicroVAX because the down-line loader is contained in boot ROM. However, you may find it useful to set the MicroVAX CPU's configuration dual inline package (DIP) switches to skip disk booting, thus enabling unattended down-line loading of the target machine. See the *MicroVAX I Owner's Manual* or *MicroVAX II Owner's Manual* for details.

A P/OS diskette is included in the distribution kit for down-line loading a system image on a VAX 8800, 8700, 8550, 8530, or 8500 processor. For the 8700, 8550, 8530, or 8500, the P/OS diskette contains the following files in the directory [CONSOLE]:

- XEBOOT.EXE — Bootstrap image for DEUNA down-line loading.
- XEABOO.COM — Sample console command procedure for booting the target processor, using the DEUNA.
- ETBOOT.EXE — Bootstrap image for DEBNT down-line loading.
- ETABOO.COM — Sample console command procedure for booting the target processor, using the DEBNT.
- DEFBOO.COM — Sample console command procedure for booting from the DEBNT. This command procedure is invoked when the BOOT command is typed, with no device specified, at the console terminal. The procedure is also invoked directly by trigger booting the system, if the system has a DEUNA, or by the command procedure NMIRESET.COM, if the system has a DEBNT.
- NMIRESET.COM — Sample console command procedure, invoked when a target system is trigger booted, if it has a DEBNT. This command procedure invokes the command procedure DEFBOO.COM.
- RESTAR.COM — Command procedure for restarting the system.

For the 8800, the diskette contains different versions of the these files in the directory [8800]. The file names have 8 appended to them. For example, the 8800 version of XEABOO.COM is XEABOO8.COM. The diskette also contains the file [8800]SECBOO8.COM for booting the secondary CPU on the 8800.

NOTE

> To boot an 8800, you must ensure that the right CPU is the primary. The sample command files ETABOO8.COM and XEABOO8.COM include the SET NEXT_PRIMARY RIGHT command before the INITIALIZE command for this purpose.

To build another copy of the P/OS diskette that contains the files just listed, you can run the command procedure NEWBOOT.COM (normally found in the VAXELN host directory ELN$:). This command procedure prompts you for the bootstrap load device, the device containing the console medium on which the loader is to be installed, and the processor type of the target machine. Answer the questions as follows:

```
$ SET DEFAULT ELN$
$ @NEWBOOT
Bootstrap device [XE]:
Console media device [CSA1]: DUAn
Processor type [730]: 8nnn
Set default bootstrap? (Y/N) [Y]:
```

where DUA*n* is the RX50 or RX33 drive and 8*nnn* is the 8800, 8700, 8550, 8530, or 8500.

Depending on your target configuration, you may have to edit one or two of the sample command files before using them. If your host system has an RX–50 or RX–33 drive, you can edit the file on your host system using a standard editor. If your host system does not have either of these drives, you can edit the file after you copy it to your P/OS disk device on the target system. See the appropriate console manual for directions on how to edit the file on the target system.

If your system has a DEBNT, you may need to edit the following line in file ETABOO.COM or ETABOO8.COM:

```
DEPOSIT R1 27        !DEBNT is on bi 2, node number 7
```

The first hexadecimal digit specifies the BI number for the DEBNT and the second hexadecimal digit specifies the node number. If your configuration differs from that, you must edit the line to match your system.

If your system has a DEUNA, you may need to edit the following line in file XEABOO.COM or XEABOO8.COM:

```
DEPOSIT R1 10       !DWBUA is on bi 1, node number 0
```

The first hexadecimal digit specifies the BI number for the DWBUA (BI-to-UNIBUS converter) and the second hexadecimal digit specifies the DWBUA node number. (The DEUNA itself is specified in the System Builder's Device Driver Menu.) If your configuration differs from that, you must edit the line to match your system.

If your system has a DEUNA, you must also edit the command file DEFBOO.COM or DEFBOO8.COM. Edit the following lines, changing ETA to XEA:

```
BOOT ETA            ! Boot from DEBNT (device type ETA), unit set in
                    !   ETABOO.COM
```

To install the supplied files on your target console system:

1. Exit the console control program.
2. Mount the P/OS diskette in a drive on your target system.
3. Copy the files to the directory BIGVOLUME:[8*nn*0], where *nn* is 50 for an 8500 or 8530, 55 for an 8550, 70 for an 8700, and 80 for an 8800, or the directory BIGVOLUME:[CONSOLE]. When you invoke a file, the system first looks on BIGVOLUME:[8*nn*0]. If the needed file is not there, the system then looks on BIGVOLUME:[CONSOLE].

   On an 8800 system, change the name of each file as you copy it, omitting the 8. For example:

   ```
   $ COPY DZ1:[8800]DEFBOO8.COM BIGVOLUME:[8800]DEFBOO.COM
   ```

4.2.2 Down-Line Loading Procedure

To down-line load a target machine, you must make the VAXELN system image file available to the network software on the host development system and have the down-line load bootstrap loader running on the target machine.

When you build the system with the System Builder, be sure to specify **Downline** as the **Boot method** entry of the System Characteristics Menu.

Use the NCP facility to store the file name of the VAXELN system image in the host system's network node data base, thus making the image's name available to the network software. For example:

```
NCP> SET NODE FRED LOAD FILE ELN$:ICP_DOWNLINE.SYS
```

The debugger can perform the same operation if you specify the EDEBUG command as follows:

```
$ EDEBUG/LOAD=ELN$:ICP_DOWNLINE.SYS
```

Start the down-line load bootstrap loader by specifying the console boot command (B) and the target machine. For example, to start the DEUNA loader on an 11/730, enter:

```
>>> B XE0
```

For an 11/750, enter:

```
>>> B DDA0
```

For a MicroVAX, enter:

```
>>> B XQA0
```

To down-line load your system image on an 8500, 8530, 8550, 8700, or 8800, enter the following boot command at the console:

```
>>> BOOT xxx
```

where *xxx* is ETA for systems with a DEBNT and XEA for systems with a DEUNA.

When the loader starts, it sends a load-request message to the host system. The network software on the host system responds by creating a Maintenance Operation Monitor (MOM) process that reads the specified VAXELN system image file and sends it to the target bootstrap loader.

When down-line loading a machine (in contrast to booting it from a disk or ROM), you do not have to set the node name or node address with the System Builder; the target machine receives its proper node name and address automatically. If you use this feature and have a system that runs on multiple processors in a network, you can use the same system image for each machine.

The following sections explain how to:

- Reload a machine that is running the Network Service
- Down-line load during debugging

- Reload production machines
- Down-line load from multiple hosts

4.2.2.1 Reloading a Machine That Is Running the Network Service

Once a VAXELN system is initialized, is running the Network Service, and has trigger booting enabled, use the remote boot command TRIGGER to boot the target machine. To use this feature, you must change the default bootstrap loader to the down-line load bootstrap loader by setting the default bootstrap selection switches to the correct read-only loader. On the 11/730, you perform this setting by using the NEWBOOT command procedure. On an 11/750, set the default boot device switch to A. On a MicroVAX, set the CPU configuration DIP switch number 1 to ON.

To trigger a target machine, use the NCP TRIGGER command. For example:

```
NCP> TRIGGER NODE FRED
```

The TRIGGER command sends a boot-request message to the target machine, which causes the VAXELN datalink device driver to halt execution of VAXELN and begin execution of the down-line load bootstrap loader (default bootstrap).

NOTE

> You can configure the DEUNA controller on an 11/750 target machine to process the boot-request message and cause the machine to halt by causing a power-failure sequence. To ensure that the 11/750 restarts, you must put the Auto Restart switch in the BOOT position. This implies that a machine requiring unattended triggering cannot also restart using memory with battery backup (that is, it will reboot when the power is restored).

If you encounter problems loading your target machine, you can use the network event-logging facility on the host system to locate the problem. To enable event logging on your host system, use the NCP SET LOGGING commands.

For example, to enable network event logging to your host's console terminal, use the following commands:

```
NCP> SET LOGGING MONITOR KNOWN EVENTS
NCP> SET LOGGING MONITOR STATE ON
```

The console displays maintenance messages and network state changes observed by the MOM network process, problems opening the VAXELN system image file, and problems communicating with the target machine.

4.2.2.2 Down-Line Loading During Debugging

During the VAXELN programming and development cycle, the target machine is likely to be down-line loaded and debugged remotely. To facilitate this operation, the VAXELN debugger can down-line load targets that are running the network and have trigger booting enabled.

The /LOAD qualifier of the EDEBUG command stores the specified VAXELN system image file name in the network node data base and triggers the target machine's down-line load bootstrap loader. In the following example, the system TEST.SYS is loaded on node FRED during an EDEBUG session:

```
$ EDEBUG/LOAD=TEST FRED
```

See Chapter 5 for a complete description of the debugger.

4.2.2.3 Reloading Production Machines

After you debug and install a VAXELN application for production use, you can continue to use the down-line loading facilities to load target machines. The host node's data base needs to contain a description of each VAXELN machine and system in the network. The description should contain the information described in the previous sections, including the file name of the production VAXELN system image file.

The default bootstrap loader on the target machines should be set to the down-line load bootstrap loader, as described previously. Whenever a target machine is rebooted (for example, after a power failure or a hardware or software crash), it must be reloaded from the host system.

4.2.2.4 Down-Line Loading from Multiple Hosts

When down-line loading target systems from multiple host systems, you can set down-line loading parameters for the target system on only one host system. If two or more hosts are capable of responding to a target system's request for down-line loading, the first to respond performs the load independently of the host that initiated the load with an NCP TRIGGER command or an EDEBUG/LOAD command.

The loading parameters are LOAD FILE, SERVICE CIRCUIT, and HARDWARE ADDRESS. If the last two parameters are set, the host attempts to load the target, even though it may not have the LOAD FILE name needed to complete the load. For example, if SERVICE CIRCUIT and HARDWARE ADDRESS are set, but LOAD FILE is not set (the typical case), the VAX/VMS MOM program still attempts to load the target system (load volunteer). This attempt blocks other hosts from loading; only later in the process will the program discover that it does not have the LOAD FILE name.

Unfortunately, this configuration error is difficult to diagnose. This error might be the problem if the system displays the following message:

```
%SYSTEM-F-TIMEOUT, Device timeout
```

If this configuration error is the problem, check the node data bases on the hosts and ensure that only one data base has the load parameters specified for the target.

For example, if you have loaded target system ABC from host XYZ and then decide to load ABC from host XXX, you should execute the following NCP command on node XYZ:

```
NCP> CLEAR NODE ABC SERVICE CIRCUIT HARDWARE ADDRESS
```

and execute the following command on node XXX:

```
NCP> SET NODE ABC SERVICE CIRCUIT UNA-0 HARDWARE ADDRESS AA-00-03-01-2B-0D
```

For a MicroVMS system, use QNA–0 instead of UNA–0 for the service circuit. For an 8800, 8700, 8550, or 8500 system, use BNA–0.

These commands ensure one node data base has the load parameters specified for the target ABC.

Chapter 5

Debugging VAXELN Systems

VAXELN provides two debuggers that you can build into your VAXELN target systems: a *remote debugger* and a *local debugger*. You can select a debugger for your system from the System Builder's System Characteristics Menu.

If you select the remote debugger, the System Builder places the nucleus of that debugger in your system. When the host development system and the target machine running your system are connected by the Ethernet, you can use the VAX/VMS command EDEBUG to access the target system and the debugging processes that are running on it.

By using the EDEBUG command, you can:

- Access one or more VAXELN target system nodes simultaneously for debugging purposes
- Use the debug symbol table information provided by VAX/VMS compilers, so you can use variable names, labels, and source-line information during a debugging session
- Let your terminal act as the console device on the target system

If you select the local debugger, the System Builder includes the entire VAXELN local debugger in your system image, creating a self-contained debugger. Having a self-contained debugger lets you enter commands at the system's hardware console terminal without having a network connection to a VAX/VMS host.

You cannot use debugger commands that require access to a source file or other host data with the local debugger. However, you can still use the local debugger to debug the VAXELN kernel and the processes on the running system.

This chapter explains:

- How to select a debugger mode with the System Builder
- The format of the EDEBUG command
- Concepts for using the debuggers
- Debugger syntax rules
- How to debug the VAXELN kernel
- The syntax and function of each debugger command

5.1 Selecting a Debugger

The VAXELN System Builder provides several alternatives for specifying how you want to debug your target system. The following options are provided on the System Characteristics Menu:

- **Local.** The System Builder includes the local debugger in the system image. You can debug the system image and the VAXELN kernel. Select this option when you want to debug the VAXELN kernel or kernel mode code running at an elevated interrupt priority level (IPL), for example, an interrupt service routine, from the target system's hardware console. Also select this option to debug kernel mode processes that run at elevated IPLs.
- **Remote.** The System Builder includes the remote debugger in the system image. You can use the EDEBUG command to remotely access the target system over the Ethernet.
- **Both.** The System Builder includes both the remote debugger and the local debugger in the system image. The remote debugger takes control for all normal system image processes. However, the local debugger can enter the kernel session at system start-up or through the SET BREAK/KERNEL command. Select **Both** if you want to debug processes within the system image from the remote debugger but also want to debug the VAXELN kernel or kernel mode code that runs at an elevated IPL.
- **None.** The System Builder does not include a debugger in the system image. Use this option after you debug your system.

From the same menu, you can select whether you want the System Builder to include the console terminal capability in the system. If you omit the console device from the system, the remote debugger makes your terminal the system's console device when you establish a connection to the node with the EDEBUG command. The System Builder automatically includes the console device when you select the local debugger.

If you select a debugger option on the System Characteristics Menu, you can specify that the debugger gains control when the job that runs a program is started. On the System Builder's Program Description Menu, select **Yes** for the **Debug** option. The debugger also gains control when each process created inside the job begins execution.

If you include a debugger in your system, the debugger also gains control of a process if a condition occurs that is not handled by an exception handler or if you explicitly ask for control by halting the process with the HALT command. The *VAXELN Run-Time Facilities Guide* contains more information on VAXELN's exception-handling mechanism.

5.2 Invoking the VAXELN Debuggers

You enter the remote debugger by issuing the EDEBUG command. You enter the local debugger in different ways, depending on the system and program characteristics defined by the System Builder. Section 5.2.1 explains the EDEBUG command. Section 5.2.2 explains how to enter the local debugger.

5.2.1 Using the EDEBUG Command

The VAX/VMS EDEBUG command, issued at the DCL prompt, lets you debug a VAXELN system you have loaded on a given node when the host development system is connected to the target machine by the Ethernet. From the **Edebug** > prompt, you can also load a new system image from the host to the target machine, start the system with or without the debugger in control, and connect to nodes that were loaded previously. Chapter 4 explains how to configure and manage the Ethernet connection.

In the following example, the EDEBUG command invokes the remote debugger to debug the system already loaded on node FRED:

```
$ EDEBUG FRED
```

After a few seconds, the following messages appear on your host terminal:

```
Edebug V3.0
Connecting to "Fred".
```

Then the initial EDEBUG screen appears (see Figure 5–1).

Figure 5–1: Initial EDEBUG Screen

```
 Job 3, process 1, program MYPROGRAM needs attention.
      Module MYPROGRAM
      3: BEGIN
      4: {this version is for debugging}
> >   5:   writeln('Flight Simulation');
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -




           Edebug V3.0
           Connecting to "Fred".
           Connected to "Fred", awaiting debug activity.
           Loading traceback data from:disk:[directory]MYPROGRAM.EXE;1
   Edebug 3,1>
```

The prompt **Edebug 3,1>** indicates that you are now in the debugger at job 3, process 1.

The debugger command DEBUG lets you debug systems on several nodes. A description of the DEBUG command is provided in Section 5.5.

5.2.1.1 EDEBUG Command Syntax

The syntax for the EDEBUG command and its qualifiers follows:

$ EDEBUG⟦/*qualifier . . .* ⟧ *node-name*

/qualifier

Descriptions of the EDEBUG command qualifiers are provided in Table 5–1.

node-name

Specifies the name of a node running a VAXELN system. Specify the node name without qualifiers if the target machine was already booted with a new system or if you are connecting to an already running target machine. Do not use the EDEBUG command for the local debugger.

If you enter the EDEBUG command without a node name, the operating system prompts you for a node:

```
$ EDEBUG RETURN
_Node:
```

5.2.1.2 Using Qualifiers to Control the EDEBUG Command

By specifying the EDEBUG command with qualifiers, you can control the way you invoke the debugger. Table 5–1 lists the EDEBUG qualifiers and their functions. The following sections provide brief explanations of how to use the qualifiers.

Table 5–1: EDEBUG Qualifiers

Qualifier	Usage
/LOAD=*system*	Loads and triggers the specified system image across the Ethernet
/NODEBUG	Exits the debugger after the specified system image is loaded and started across the Ethernet

5.2.1.2.1 Loading a System Across the Ethernet (/LOAD)

With the EDEBUG command, you can use the /LOAD qualifier to load and start a system image across the Ethernet. Specify the /LOAD qualifier with a system image file (type SYS) that includes the remote debugger. For example, the following command loads and starts the system image MYSYSTEM.SYS on the target machine FRED:

```
$ EDEBUG/LOAD=MYSYSTEM FRED
```

When you use the /LOAD qualifier, the new system replaces the system that is already running on the target machine.

5.2.1.2.2 Exiting from the Remote Debugger (/NODEBUG)

To exit the debugger immediately after you load and start a system image across the Ethernet, specify the /NODEBUG qualifier. For example:

```
$ EDEBUG/NODEBUG/LOAD=MYSYSTEM FRED
```

This command loads and starts the system image MYSYSTEM.SYS on the target machine FRED, then exits the debugger.

After you exit the debugger, the target system on that node is left in a suspended state, if any jobs or processes are built with debugger control. Jobs and processes that are not controlled by the debugger continue to run. Jobs and processes that are built with debugger control, however, remain suspended until you connect to the node, return control to the debugger, and issue debugger commands to allow them to begin executing.

5.2.2 Entering the Local Debugger and the Kernel Session

When a job under local debugger control is created, the local debugger prompt appears at the target console. For example:

```
Edebug 3,1>
```

While you are debugging your system image, you might need to set breakpoints and examine locations in the VAXELN kernel image, in interrupt service routines, or in other kernel mode code at elevated IPLs. If you included the local debugger in your system image, you can perform these debugging operations by entering commands from the target hardware console terminal.

To enter the kernel session, do one of the following:

- Type the following on the hardware console when you boot the system:

  ```
  >>> B/R5:4 device
  ```

  This command activates the kernel session during the system's initialization sequence. If your target processor is an 8800, 8700, 8550, 8530, or 8500, you can instead edit the sample boot command file (XEABOO.COM or ETABOO.COM) to deposit 4 into R5. Remove the comment indicator (!) from the line:

  ```
  ! DEPOSIT R5 4
  ```

- Place the target machine in hardware console mode (see the appropriate hardware manual for the correct procedure), then type:

  ```
  >>> D/I 14 5
  >>> C
  ```

  This command activates the kernel session if the system is not executing above interrupt priority level 5 (hexadecimal). If the system is executing at or above that interrupt priority level, you cannot get control at the console.

- Type the following from the local debugger:

  ```
  Edebug 4,5> SET SESSION/KERNEL
  ```

  This command instructs the local debugger to attach the session associated with the kernel.

- Issue a SET BREAK/KERNEL command from the local debugger or, if both debuggers are included, the remote debugger. Use this method for debugging interrupt service routines or kernel mode jobs running at elevated IPLs (see the SET BREAK command).

The following prompt appears during the kernel session:

```
Kernel Edebug>
```

To exit the kernel debugging session, use the GO command.

5.3 Using the VAXELN Debuggers

The VAXELN debuggers let you examine or deposit memory locations, evaluate expressions, set breakpoints, and control the execution of your programs and system. The debuggers also let you perform systemwide operations, such as displaying the jobs running on the system.

You can create VAXELN applications that consist of multiple jobs and processes executing on several nodes in a network. Therefore, when you debug a system, you might need to control more than one job or process and more than one target system at the same time. The VAXELN debuggers have features that handle this requirement. For each process that you debug, the debugger establishes a *command session* (see Section 5.3.1). The debugger directs the commands that you enter to the process associated with the command session. You can change the command session within a node with the SET SESSION command. For processes on different nodes, use the DEBUG command to change nodes and then use the SET SESSION command to select a process. The DEBUG command changes the command session to the first process waiting for attention on the new node.

A command session can be in one of the following states:

- The session's process is not waiting for a debugger command.
- The session is suspended, waiting for a debugger command.

If the state of a session changes (for example, if a breakpoint is encountered), you are immediately notified, even if that session is not the current command session. A process whose session is waiting for debugger commands remains inactive while you work with the command session.

If no process is running under debugger control, you can enter the debugger through a Control-C session (see Section 5.3.3). You can also use the Control-C session to change a command session before a breakpoint occurs.

5.3.1 Process Identifiers

To identify a command session, the debugger uses a *process identifier* for each process in a running system. A process identifier lets you activate a process's command session.

A process identifier consists of three parts:

- **Job** — The job name or the identification number (ID) assigned by the kernel when the job is created. The name is usually the name of the job's program, as specified through the System Builder or the debugger's LOAD command. If you have more than one instance of a program in a system, you can identify the instance with the job ID or with the job name and the version number found on the system map (see Section 3.4.5). Use the SHOW SYSTEM debugger command to find the IDs assigned to the jobs in the system.
- **Process** — The identification number assigned by the kernel when the process is created. The SHOW PROCESS debugger command displays this number.
- **Node** — The node on which the process is running (remote debugger only).

To specify a process identifier, use the following format:

job,⟦*process* ⟧ ⟦ *node*⟧

If you omit the process, process number 1, the job's master process, is assumed. If you omit the node, the current session's node is assumed. Some examples follow:

```
4,3
CONSOLE,3
CONSOLE,3 NODE10
```

The first example consists of a job number and process number. The second example consists of a job name and process number. The third example consists of a job name, process number, and node name.

Nonalphanumeric characters are not allowed in debugger syntax. Therefore, if a job name contains nonalphanumeric characters, you must enclose the job name in single quotes. For example:

```
'MYJOB;2',3
```

5.3.2 Command Sessions

The identifier for the process whose command session is active is included in the debugger prompt. For example:

```
Edebug 4,5>
```

The preceding prompt indicates that the commands you type are directed to process 5 in job 4. If you are debugging more than one node, the node name is also included in the prompt.

The command session is set when the debugger is invoked for the first process waiting for debugger attention. The command session remains constant until you change it with the DEBUG or SET SESSION command or until the command session's process exits. The session also changes if another session hits a breakpoint while the current session is in a running state. When the command session's process exits, the session is reset to the job's master process, or if the current process is the master process, the debugger selects a session that is waiting for command input. If no process is waiting for a command or there is no current session, the debugger prompt does not appear. In that case, you can use the Control-C session to activate a command session.

The debugger does not prompt for commands while a command session's process is in a running state.

5.3.3 Control-C Session

If you need to activate a command session when no debugging session exists or when the process of the active session is running, you can activate a *Control-C session.* A Control-C session lets you enter commands that are not directed at a particular process or lets you switch to another session. For example, you might want to enter the command SHOW SYSTEM, which displays the jobs in the system.

Activate a Control-C session by typing CTRL/C:

```
[CTRL/C]
EDEBUG CONTROL-C>
```

The prompt indicates that a Control-C session is active. To return to the state that was active when you typed CTRL/C, enter a null command line. If you type CTRL/C again, the executing command aborts and you return to a command session prompt.

When a Control-C session is active, you cannot use some of the debugger commands. For example, you cannot use the EXAMINE or the DEPOSIT command to examine or deposit memory, because no process context is associated with the session. You can, however, use the following commands:

- The DEBUG command
- The EVALUATE command
- The HALT command
- The SET SESSION command
- Systemwide commands

5.3.4 Breakpoints

If a program is specified for debugger control, the debugger gains control when the job and its master process are created. At the beginning of this command session, execution of the associated program stops, so you can enter debugger commands. The debugger also gains control whenever a new process within the job is created, unless you use the CANCEL CONTROL command. Execution continues after you enter the GO or STEP command. The GO command causes the system to continue processing until a breakpoint occurs. The STEP command lets you execute the program one line or instruction at a time.

A *breakpoint* is similar to a stop sign. You can set breakpoints that affect only the current session's process, or you can set breakpoints that affect an entire job. You place a breakpoint at a location in your program, and when that breakpoint is reached during execution, the program stops running. For example, when the debugger is in control of the current session, if you set a breakpoint and type GO, the program executes until it reaches the breakpoint; then it stops. You can then examine variables and perform other operations in the current context.

NOTE

> A breakpoint set for a specific process does not stop program execution if the code in that process is executed by another executing process.

To set breakpoints, use the SET BREAK command. For example:

```
Edebug 4,5> SET BREAK 500
```

The SET BREAK command also lets you specify a command that is to be executed when the program stops at the breakpoint.

To continue program execution from the point where the breakpoint occurred, type the GO or STEP command.

For more information about the SET BREAK command, see the command description in Section 5.5.

5.3.5 Using the Remote Debugger

The following sections discuss features available only with the remote debugger.

5.3.5.1 Symbolic Debugging

If you are using the remote debugger, you can debug your system symbolically — that is, you can refer to program locations, variables, or constants by name, and you can view source lines. The debugger uses information provided by the VAX compilers and linker in the object (OBJ) and program image (EXE) files.

You request the symbolic debugging feature by specifying the /DEBUG qualifier at compile and link time (see Chapter 2) and by selecting **Remote** or **Both** for the **Debug** option on the System Builder's Program Characteristics Menu. A debug symbol table is generated, which contains the following information for each module that contributes to the program image:

- Information that relates addresses to the line numbers in a module
- Information that relates addresses to the source line that produced the code at those addresses
- Symbolic information (names and types) about the module's variables

The debug symbol table contains the information about variables only if you requested it when you compiled and linked the module.

NOTE

Complete debug symbol tables are quite large, and they make a program image much larger. Therefore, after you debug your programs, recompile them without requesting the debug symbol table. Source-line and traceback information is comparatively small and can be left in the image. This information is included in compilations by default.

The debug symbol table is associated with all the processes in the job running a program. The association is either implicit (when a job starts under debugger control) or explicit (by use of the debugger's SET PROGRAM command). For more information about the relation between the symbol table and references in the debugger, see Section 5.4.2.

5.3.5.2 Command Files

You can create command files containing debugger commands. To execute the command file, specify the @ command with the name of the command file. For example:

```
@DEBUGCOMMANDS
```

The commands in the file DEBUGCOMMANDS.COM execute. You can nest the command files eight levels deep.

The debugger uses the logical names DBG$INPUT and DBG$OUTPUT for I/O; these names are usually assigned to SYS$INPUT and SYS$OUTPUT, respectively. You can use these logical names in debugger command files.

5.4 Debugger Syntax Rules

The VAXELN debugger command language is designed to be simple to use for programmers who are familiar with several programming languages. The VAXELN debugger command language is generic to most programming languages and is similar to the command language for the VAX/VMS debugger, DEBUG-32. The DEBUG-32 command language provides a fully functional command language that is dynamically tailored to the language in use at the point where the program is stopped.

The VAXELN debugger command syntax and semantics are modeled after the DEBUG-32 command language, and wherever possible, a command performs the same function in both debuggers.

The syntax for the debugger commands follows:

command ⟦*/qualifier . . .* ⟧ *parameter . . .*

The qualifiers and parameters you can specify with each command are described in the individual command descriptions (see Section 5.5).

NOTE

> Qualifiers must immediately follow the command and must precede the parameters.

The following example shows how to enter the debugger command EXIT:

```
Edebug 4,5> EXIT
```

The EXIT command terminates the debugger.

The next example shows how to enter a command with a qualifier and a parameter:

```
Edebug 4,5> EXAMINE/HEX my_variable
```

This command examines the variable *my_variable* and displays the value in hexadecimal form.

If a command does not conform to the command format or if the debugger does not recognize the command name, the debugger treats the command as an expression and evaluates it. For example:

```
Edebug 4,5> 123+456
```

This command displays the value 579. This evaluation feature also lets you examine the value of a variable by typing its name. For example:

```
Edebug 4,5> x
```

This command displays the value of the variable *x*.

NOTE

> In this chapter command and qualifier names are sometimes abbreviated to their shortest unique form.

5.4.1 Expressions

Many of the debugger commands must be specified with arithmetic expressions. These expressions are algebraic sequences consisting of constants, parentheses, operators, and variable references.

The syntax of these arithmetic expressions does not parallel that of any language in particular. The semantic rules for interpretation follow the normal rules for algebraic expressions.

Arithmetic expressions you specify with debugger commands can include the following arithmetic, Boolean, and address-related operators:

Arithmetic Operators:

+
–
*
/
DIV
MOD
@

The + and – operators can be either sign (positive or negative) or arithmetic (plus or minus) operators.

Boolean Operators:

=
<>
<
<=
>
>=
AND
OR
NOT

Address-Related Operators:

ADDRESS(*variable*)
@

The arithmetic operator @ specifies an arithmetic left-shift if the count is positive or a right-shift if the count is negative. For example:

```
REF @ 2
```

This command shifts the contents of REF two bits to the left.

You can use the Boolean operators = and := interchangeably to express assignments.

Use the address-related operator @ to get the value of a location represented by an address expression. Use the ADDRESS function to get the address of an addressable variable. Some variables, such as those assigned to registers, are not addressable.

5.4.1.1 String Expressions

String expressions consist of one or more string values separated with the + operator. The operator joins the values into one string value. For example:

```
'HEADER'+STR1
```

5.4.1.2 Address Expressions

Address expressions return the address of a location, while other types of expressions return the contents of a location. To understand the difference, consider the following Pascal assignment statement, in which *a*, *b*, and *c* are variables:

```
a := b+c;
```

Variables *b*, *c*, and *b+c* represent values to be assigned to variable *a*. Variable *a* represents an address expression designating the address to receive the value. In the debuggers, the left side must be an expression for a valid address. If *a* were on the right side of the assignment operator, it too would represent a value, not an address.

The following command examines the contents of address 1234:

```
EXAMINE 1234
```

The following command examines the contents of the address represented by the variable *myvar*:

```
EXAMINE myvar
```

The following command examines the contents of the address represented by the variable *myvar*+2:

```
EXAMINE myvar+2
```

Address expressions consist of variable references, integer constants, and the dyadic operators –, +, *, DIV, /, and @. Parentheses are allowed. You can also include the monadic operators + and – in address expressions. The @ operator implies the contents of the specified address. For example, the following command displays the contents of the program counter (an address):

```
EXAMINE PC
```

To display the instruction at the PC address, specify:

```
EXAMINE/INSTRUCTION @PC
```

Since the PC contains addresses, not instructions, the following command is invalid:

```
EXAMINE/INSTRUCTION PC
```

The following command is invalid because the identifier PC refers to the program counter itself, not to its contents.

```
SET BREAK PC
```

To set a breakpoint at the address contained by the program counter, you should specify the command as follows:

```
SET BREAK @PC
```

The address-versus-contents distinction also applies to symbolic references to pointer variables. In the following example, *ptr* is a pointer variable:

```
EXAMINE ptr
```

The preceding command examines the contents (an address) of *ptr*. The following example, however, examines the memory pointed to by *ptr*:

```
EXAMINE @ptr
```

NOTE

Equivalent forms of the @ operator are: *ptr^*, **ptr*, and *ptr*→.

When symbols defined with the DEFINE command appear in address expressions, the symbols yield values, much as constants do. In these cases, the symbols are similar to registers because they always express a value.

5.4.2 Identifiers

During a debug session, you can refer to items by their text name, or identifier. You can use three types of identifiers:

- Identifiers defined with the DEFINE command
- Predefined identifiers
- Program locations and variable names available to the debugger that are defined by a program's debug symbol table entries

Identifiers are names consisting of up to 31 ASCII alphanumeric characters, the dollar sign ($) character, or the underscore (_) character. An identifier can consist of uppercase and lowercase characters; however, an identifier cannot begin with a number.

5.4.2.1 Defining Identifiers

The local debugger and the remote debugger let you use the DEFINE command to define identifiers for use in expressions and commands. These identifiers can represent variables, constants, or addresses. For example:

```
Edebug 4,5> DEFINE myval = 40
```

The following command defines the debugger variable *an_integer* with an INTEGER data type and assigns that variable the value 10. The identifier *an_integer* then yields 10 when you specify it.

```
Edebug 4,5> DEFINE an_integer :: INTEGER = 10
```

Table 5–2 lists the data types you can specify with the DEFINE command. The default DEFINE data type is INTEGER.

Table 5–2: DEFINE Data Types

Data Type	Description
INTEGER	Simple integer
BOOLEAN	TRUE or FALSE
BYTE_DATA(*n*)	Bytes of unspecified data
REAL or FLOAT	Single-precision floating-point
DOUBLE or GRAND	Double-precision floating-point

Table 5–2 (Cont.): DEFINE Data Types

Data Type	Description
HUGE	Double double-precision floating-point
STRING(*n*)	Character string with fixed size of *n* characters
CHAR	Character string with a size of one character
VARYING_STRING(*n*)	Character string with varying size of up to *n* characters
RELOCATION	Relocation constant

You can specify an expression for *n* when you use the keywords BYTE_DATA(*n*), STRING(*n*), and VARYING_STRING(*n*). For example:

```
Edebug 4,5> DEFINE my_string :: STRING(x+3)
```

If you do not specify a value for *n*, 1 is assumed.

The data type RELOCATION is similar to INTEGER, but the value represents program locations. You can use this data type anytime; however, it is most useful when symbolic debugging is not available. A display of a program location (as with the SHOW SESSION command) shows the location relative to the already defined RELOCATION data item whose value is closest to the location, if the distance does not exceed 2048 bytes. For example, the following command defines a RELOCATION constant named *mbase*:

```
DEFINE mbase :: RELOCATION = 2000
```

When an address to be displayed is within 2048 bytes of *mbase*, the address is displayed as *mbase+byte_offset*. For example, if the address is 2020, the debugger displays it as *mbase+20*.

You can encounter a debugger-defined variable with the same name as a variable in your program. To distinguish debugger-defined identifiers, prefix them with a percent sign (%). This character tells the debugger to use the internal identifier, not the program variable.

For example, *my_integer* could be declared a variable in a Pascal program by the following command:

```
VAR my_integer: INTEGER;
```

my_integer could be defined as a debugger internal identifier by the following command:

```
DEFINE my_integer
```

The following strings refer, respectively, to these uses of *my_integer*:

```
my_integer
%my_integer
```

You can change the value of a defined identifier by using another DEFINE command or by using the DEPOSIT command. For example:

```
Edebug 4,5> DEPOSIT an_integer = 20
```

This command changes the value of the variable *an_integer* to 20.

See Section 5.5 for a complete description of the DEFINE command.

5.4.2.2 Predefined Identifiers

The debuggers contain several predefined identifiers, including identifiers that let you access fixed hardware entities.

For example, the following general-purpose registers are predefined:

```
R0, R1, R2, . . . , R11, AP, FP, SP, PC
```

Other predefined identifiers include:

PSL	Processor status longword
$	The constant 80000000 (hexadecimal)
NIL	The constant 0
LABEL *string*	The location of the label *string*
LINE *n*	Line *n* of a program

If you encounter a conflict with a program's symbol, prefix the predefined identifier with a percent sign (%). This character tells the debugger to use the internal identifier, not the program variable.

When debugging a program that runs in kernel mode or within the kernel debugging session, the debugger names the internal processor registers P*n*, where *n* is a decimal integer. For example, P1 is the executive-mode stack pointer.

5.4.2.3 Program Locations and Variable Names

When you use the remote debugger, your program image can provide the debugger with a debug symbol table. You can use this symbol table to refer to locations and variables by the names you used within the program.

The syntax for expressing program locations and variables that are included in the debug symbol table follows:

⟦*path-name*⟧*variable-reference*

The variable reference is the name of a variable or location you used in your program. The path name qualifies the variable reference to a particular module or routine in the program.

The debugger makes assumptions about the path name, letting you leave out the path-name parameter in most cases. The default path name is referred to as the *reference scope* or *view scope*. This scope is established by the debugger, based on where your session stops. For example, if your program stops in a routine named INITIALIZE contained in a module called DRIVER, the view scope is set to duplicate the compiler's view of the variable reference scope for the routine INITIALIZE. You can refer to variables and locations inside the routine without a path name.

However, you must specify a path name when you want to refer to a name that is not visible inside the routine where the session stops; for example, if the session stops in another module.

You can specify path names as follows:

module-name⟦\routine-name\\ . . . ⟧

The *module-name* parameter is the module in which the variable is defined. You can use the optional list of routine names to qualify the path name if the variable is internal to some routine. For example, the path name for routine INITIALIZE in module DRIVER is:

```
driver\initialize\
```

In some cases, you need to specify only the module name. In other cases, you must specify the sequence of routine names that identify the area of the program in which you are interested. An example of a path name for the variable *a* is:

```
file_io\open_routine\a
```

You can use predefined identifiers in path names to specify locations within a program. For example:

```
declare\%LINE 10
```

This path name uses the predefined identifier LINE to refer to line 10 in module *declare*. You can omit the module name and backslash if the debugger is stopped at a point in module *declare*.

NOTE

> In highly optimized programs, the compiler might have eliminated code at a particular location, even though the source module has explicit statements there. Optimizing may also cause the compiler to eliminate variables.

You can also use the predefined identifier LABEL to refer to the location of a label in a program. For example:

```
outmodule\%LABEL errormessage
```

Again, you can omit the module name and backslash if the label is defined in the view scope.

The debugger assumes that the view scope is associated with the program that is executing when your session stops. Since this location might not be convenient, you can change the view scope by moving back and forth along the call history stack. To move along the stack, use the PREDECESSOR and SUCCESSOR commands. PREDECESSOR moves the view scope back to the caller of the routine in which your session stops. SUCCESSOR moves the view scope forward.

When the stopping point differs from the view scope, the debugger displays both the view scope and the place where your session stops. When your program stops in a location that is not part of the known program, the debugger searches back on the call history stack until it finds the last active place in the program.

5.4.3 Variable References

The debugger provides a generic syntax for specifying variables. Examples of simple variable names follow:

i
my_variable
his_variable

You can specify an entire structure or individual variables within the structure. For example:

rec1
rec1.item2
rec1.item3.value1

You can express array variables in two notations:

array1(1,2,3)
array1[1,2,3]

You can also specify entire arrays: *array1*

You can specify pointer-qualified variables in the following forms:

*ptr_variable^*
ptr_variable→
**ptr_variable*
@ptr_variable

5.4.4 Types and Typecasting

Each expression, value, or variable reference in a debugger command line has an intrinsic data type, which affects the item's memory size, display format, and expression semantics. These data types are used explicitly in several contexts within the debugger command language. See Table 5–2 for a list of the available data types.

You can use data types as qualifiers with the EXAMINE and DEPOSIT commands to specify a data type to use in the display or deposit of a value. For example:

```
Edebug 4,5> EXAMINE/REAL counter
```

This command causes *counter* to be examined and displayed as a floating-point value regardless of its true data type. Here the item's intrinsic memory size is also being specified; therefore, the debugger examines four bytes regardless of the size of *counter*. For more information, see the descriptions of the EXAMINE and DEPOSIT commands in Section 5.5.

You can specify data type names as part of a variable reference to *cast* the reference's intrinsic data-type-to-variable reference. Use this feature inside expressions to force particular semantic interpretations. For example:

```
Edebug 4,5> EVALUATE gas_v >= counter :: REAL
```

This command evaluates the Boolean expression that compares the real variable *gas_v* with the integer variable *counter*. However, it is necessary to interpret *counter* as a real number rather than as its implied integer type. This syntax is called *typecasting*. You specify typecasting by using double colons (::) followed by a type specifier.

Consider the following example:

```
Edebug 4,5> EXAMINE/REAL counter
```

This command uses the REAL data type as a qualifier to the EXAMINE command. The following typecasting example has the same effect:

```
Edebug 4,5> EXAMINE counter :: REAL
```

The debugger does not attempt to understand all data types available in all programming languages. Rather, it understands and operates on several basic computational types. When interpreting the information in a program's debugger symbol table, the debugger translates the language-specific data type into one of its generic data types. You can determine the debugger's interpretation of a variable by using the SHOW SYMBOL command. The data types used by the debugger are defined in Table 5-2.

Some of these type names (STRING, for example) do not have an associated static size. In these cases, it may be necessary to specify a size value. For example:

```
STRING(10)

BYTE_DATA(100)
```

You can specify the size value as an expression, which is interpreted when you use the type specifier. For example:

```
Edebug 4,5> EXAMINE d_block :: STRING(50 div ctr)
```

If you do not specify a size value, 1 is assumed.

5.4.5 Computational Constants

This section defines the syntax rules for the computational constants supported by the VAXELN debuggers.

5.4.5.1 Boolean Constants

Boolean constants are the reserved identifiers TRUE and FALSE, denoting the numeric values 1 and 0, respectively.

5.4.5.2 Integer Constants

Integer constants are strings of characters beginning with a digit from 0 to 9. Numbers are interpreted in the current radix. For instance, 1234 or 1FEF4 are examples of integer-valued constants. The radix in use for a particular debugging session is dictated by the place in the program where the session stops, and the explicit settings of the radix are done with the SET MODE command.

You can explicitly specify a radix by prefixing the numerical string with a % construct. The explicit radix specifiers are:

%X or %HEX for hexadecimal
%D or %DEC for decimal
%O or %OCT for octal
%B or %BIN for binary

For example:

```
%O177440
```

If you want to include spaces in a numerical string, enclose the string in single quotes. For example:

```
%X'ff ff ff ff'
```

When the default radix is hexadecimal, the debugger interprets unrecognized identifiers as hexadecimal numbers. This feature helps avoid the need to specify %X. For example, the following are valid hexadecimal constants:

```
FE00
0FE00
%XFE00
```

Integer constants must always be in the range -2^{31} to $2^{31}-1$ or in the range 0 to $2^{32}-1$.

5.4.5.3 Floating-Point Constants

If the default radix is decimal, specify floating-point constants in FORTRAN format. For example:

```
1.24
1.2e10

1.234e-10
-12.4
```

If the default radix is not decimal, you must use the %F construct to specify floating-point constants. For example:

```
%F'1.24'
%F'1.2e10'

%F'1.234e-10'
%F'-12.4'
```

The numbers in quotes are interpreted as decimal numbers.

Floating-point constants are always converted to the VAX double-precision format. Depending on the model of the VAX machine you are using, the range of floating-point numbers varies. Normally, the debugger uses the VAX DOUBLE floating-point format, providing for approximately 16 digits in the range .29e–38 to 1.7e38.

On some VAX machine models, the data type GRAND is available. This data type is another double-precision floating-point data type that has a smaller degree of precision but a larger exponent range, providing for approximately 15 digits in the range .56e–308 to .9e308.

You can change the default floating-constant type with the SET MODE command.

5.4.5.4 String Constants

String and character constants are sequences of characters enclosed in apostrophes. A pair of consecutive single quotes ('') inside a string constant represents a single apostrophe character. String constants in the debugger can have up to 132 characters. A string constant cannot span multiple lines. An example of a string constant follows:

```
'This is a string.'
```

5.4.5.5 Special Constants

In address expressions, a period (.) is shorthand for the address of the source for the last EXAMINE operation or for the target of the last DEPOSIT operation.

Some items you can examine are not addressable, in which case the debugger issues an error message saying that the period is meaningless. For example, variables that are not stored on byte boundaries are not addressable.

A backslash (\) represents the value of the most recent expression.

RELOCATION is a data type keyword identifying a relocation constant whose value represents a program location (see Section 5.4.2).

5.4.6 Comments

You can include comments in command lines by preceding the comment text with an exclamation point (!). For example:

```
Edebug 4,5> EXAMINE/REAL counter  ! This is a comment
```

5.5 Command Summary

The debugger command syntax is the same for the remote and local debuggers. However, you can do symbolic debugging and display source lines only if you use the remote debugger; otherwise, you cannot access the debug symbol table and the program's source text.

The debuggers inform you if a particular command is invalid. For example, you cannot use the EXAMINE/SOURCE command if you are using the local debugger.

The rest of this section contains descriptions of the commands you can use to debug VAXELN systems.

CALL

Calls a routine in the context of the current program and returns to command mode when the routine completes.

NOTE

If the routine affects the state of the running program, the debugger might lose control of the session. In addition, the debugger does not handle breakpoints or exceptions while the routine is executing.

Format

CALL *target ⟦argument . . . ⟧*

Parameters

target
A path-name identifier, integer, or address expression.

Specify an address expression when the target's name is not a simple identifier. You must enclose the address expression in parentheses.

argument
One or more optional arguments to the routine being called. If you specify multiple arguments, separate them with commas and enclose the list in parentheses. You can specify up to 16 arguments. After evaluation, each argument must fit in one longword (32 bits).

Description

Use the CALL command when the debugger is unable to perform a complex function. A typical example might be in debugging a command interpreter, where the CALL command calls a routine in your application, which dumps the debugger symbol table.

After the routine executes, the debugger displays the value returned by the called routine in register R0.

For example:

```
Edebug 4,5> CALL dump_symbol (@R0)
Symbol table entry at: 43450  Name: test1 Type: integer  Value: 2
16 (00000010)
```

Examples

1.
```
Edebug 4,5> CALL %X200
16 (00000010)
```

2.
```
Edebug 4,5> CALL external_call (1,0,FALSE)
(Parameters in HEX are 00000001 00000000 00)
1 (00000001)
```

3.
```
Edebug 4,5> CALL (rtn_ptr+0) (1,0,FALSE)
2 (00000002)
```

CANCEL BREAK

Cancels breakpoints.

Format

CANCEL BREAK *⟦address⟧*

/ALL
/KERNEL

Parameter

address
An expression for the address at which a breakpoint is to be canceled. This parameter is optional. If you omit it, specify the /ALL qualifier. If you specify an address for which a breakpoint is not set, the system displays the following error message:

```
%EDEBUG-E-NO_SUCH_BREAKPOINT, specified breakpoint was not set
```

The SHOW BREAK command displays the current breakpoints.

Qualifiers

/ALL
Cancels all breakpoints.

/KERNEL
Cancels breakpoints set by the SET BREAK/KERNEL command.

Example

See the SET BREAK command description.

CANCEL CONTROL

Causes all processes that start in the command session's job to start executing independently of the debugger. The debugger does not take control when the process is created. To reverse this action, use the SET CONTROL command.

Format

CANCEL CONTROL

CANCEL EXCEPTION BREAK

Reestablishes the default exception handler's search for the associated session, reversing the action of the SET EXCEPTION BREAK command.

Format **CANCEL EXCEPTION BREAK**

CREATE JOB

Creates a job in the target system that runs a designated program.

Format

CREATE JOB *program-name ⟦(argument . . .)⟧*

Parameters

program-name
A string expression that names a program already installed in the system. Use the SHOW PROGRAM/ALL command to display the names of all installed programs along with other information about them.

argument
One or more optional arguments to the program enclosed in parentheses. If you specify multiple arguments, separate them with commas. You can specify up to 16 arguments. Program arguments are strings. The strings must be constants, DEFINE values, variables, or strings explicitly typed in.

Example

```
Edebug 4,5> CREATE JOB TSTJOB ('1','2','3')
```

CREATE PROCESS

Calls a routine declared as a process block in the current program and starts the procedure as a process.

Format

CREATE PROCESS *target* ⟦*(argument . . .)*⟧

Parameters

target
A path-name identifier, integer, or address expression.

Specify an address expression when the target's name is not a simple identifier. You must enclose the address expression in parentheses.

argument
One or more optional arguments to the routine enclosed in parentheses. If you specify multiple arguments, separate them with commas. You can specify up to 16 arguments. The arguments must be specified in the format that matches the declaration in the process block header. After evaluation, each argument must fit in one longword (32 bits).

Description

The debugger gets control of the new process when it is ready to begin executing. If you have issued a CANCEL CONTROL command before creating the process, you will get no response if the process starts successfully. If an error occurs, a message is displayed to indicate the failure status.

CTRL/C

Aborts the operation in progress and, except when in the kernel session, invokes the debugger's command interpreter.

Format

CTRL/C

CTRL/Z

Exits the debugging session.

Format CTRL/Z

DEBUG

Debugs, connects to, or loads and debugs a system on another node; changes the command session to the first process waiting for attention on the new node; can also trigger the boot operation on that system. Once the name of a node is specified with the DEBUG command, that node is known to the debugger for the duration of the debugger session.

The DEBUG command is not valid for the local debugger during the kernel session.

Format

DEBUG *node*

/LOAD=*system*

Parameter

node
If the /LOAD qualifier is not specified, the name of a node that is running a VAXELN system; if the /LOAD qualifier is specified, the name of the target node that is to be loaded.

Qualifier

/LOAD=system
Loads the specified system image on the specified node.

Example

```
Edebug 4,5> DEBUG FRED
```

DEFINE

Creates or redefines a session variable with a specified type and an initial value.

Format

DEFINE *identifier ⟦:: type⟧ ⟦:= expression⟧*

Parameters

identifier
The session variable that is to be created or redefined.

type
A valid debugger data type (see Table 5–2). This parameter is optional, and the default is INTEGER. The colon (:) after the *type* parameter is optional.

expression
A valid debugger expression to indicate the initial value. This parameter is optional, and the default is 0. You can redefine session variables at any time.

Description

Variables defined this way are available in all sessions. If a program is associated with a session, a reference to one of these identifiers is resolved to the session variable only after the reference scope is searched. If the name is prefixed with a percent sign (%), however, it refers unambiguously to the session variable.

You cannot redefine the symbols R*n*, P*n*, PSL, SP, AP, FP, or PC. When used in expressions, these symbols yield the contents of locations and cannot be used as address expressions.

This command is very useful in the local debugger, where source files are not available. You can define a session variable of the RELOCATION type for the start of a module. You can then refer to locations within the module as offsets in your module listing from the start of the module.

Examples

1.
```
Edebug 4,5> DEFINE iii
Edebug 4,5> EVALUATE iii
iii:  0    (00000000)
Edebug 4,5> DEFINE iii = 2
Edebug 4,5> EVALUATE iii
iii:  2    (00000002)
```

2.
```
Edebug 4,5> DEFINE ptr1 :: ^INTEGER := NIL
```

3.
```
Edebug 4,5> DEFINE s1 :: STRING(10) = ''
```

4.
```
Edebug 4,5> DEFINE bd1 :: BYTE_DATA(30) = S1
```

5.
```
Edebug 4,5> DEFINE rtn_ptr :: RELOCATION
Edebug 4,5> DEPOSIT rtn_ptr = ADDRESS(external_call)
Edebug 4,5> EXAMINE/INSTRUCTION rtn_ptr+2
rtn_ptr+2: MOVAB -40(FP),SP
Edebug 4,5> DEFINE relo :: RELOCATION = %X200
Edebug 4,5> EXAMINE/BYTE:40 relo
18000300 00000010 00000004 00003039 | relo + 0000 | 90..............
00000000 00000000 00000000 00000014 | relo + 0010 | ................
                  00000000 00000061 | relo + 0020 | a.......
```

6.
```
Edebug 4,5> DEFINE module_base :: RELOCATION = %X2546
Edebug 4,5> SET BREAK module_base+%X176
Edebug 4,5> EXAMINE/INSTRUCTION module_base+%X176
```

DELETE PROCESS

Deletes a process associated with a session.

NOTE

You can only use this command to delete processes associated with sessions that are stopped. If necessary, first use the HALT command to stop the session.

Format

DELETE PROCESS *⟦process⟧ ⟦node⟧*

Parameters

process
The process identifier of the process to be deleted session. This parameter is optional. If you do not specify a process, the command deletes the process associated with the current session.

node
The name of the node on which the process being deleted resides. This parameter is optional. If you omit it, the current node is assumed.

DEPOSIT

Deposits the value of an expression in a location described by a variable reference or address expression.

The qualifiers specify the amount of storage examined, overriding the size associated with the item's type. If the command has no associated data type, INTEGER (longword) is the default.

Format

DEPOSIT *address := expression*

/ASCII:*size*
/BYTE:*size*
/DOUBLE
/D_FLOAT
/FLOAT
/G_FLOAT
/GRAND
/H_FLOAT
/HUGE
/LONGWORD
/QUADWORD
/REAL
/WORD

Parameters

address
An expression for the address at which a value is to be deposited. If you specify a complex expression, enclose it in parentheses. You can use the = symbol interchangeably with the := symbol.

expression
The expression whose value is to be deposited at the specified location.

Qualifiers

/ASCII
Specifies that the target is a string.

`/ASCII:`***`size`***

Specifies that the target is a string of a given size. The size specifier can be:

> *:integer*
> *=integer*
> *:(expression)*
> *=(expression)*

The default size is 1.

/BYTE
Specifies that the target is an arbitrary sequence of bytes.

`/BYTE:`***`size`***

Specifies that the target is a byte sequence of a given size. The size specifier can be:

> *:integer*
> *=integer*
> *:(expression)*
> *=(expression)*

The default size is 1.

/D_FLOAT
Specifies that the target is a D_floating value.

/DOUBLE
Specifies that the target is a D_floating value.

/FLOAT
Specifies that the target is an F_floating value.

/G_FLOAT
Specifies that the target is a G_floating value.

/GRAND
Specifies that the target is a G_floating value.

/H_FLOAT
Specifies that the target is an H_floating value.

/HUGE
Specifies that the target is an H_floating value.

/LONGWORD
Specifies that the target is an integer.

/QUADWORD
Specifies that the target is a quadword.

/REAL
Specifies that the target is an F_floating value.

/WORD
Specifies that the target is an integer.

Description

Some implicit conversion is performed for you at deposit time, and if the conversion cannot be done, an error is signaled. The rules for conversion are:

1. If the target is an integer of Boolean value, then the source must be integer or Boolean.
2. If the target is floating, then the source must be integer or floating. The integer value is converted to floating.
3. If the target is string, then the source must be string or byte data. The source is padded at the end with blanks or truncated at the end to fit the target.
4. If the target is byte data, then the source must be string or byte data. The source is padded with zeros or truncated to fit the target.

Examples

1.
```
Edebug 4,5> DEPOSIT 123+456 = 10
```

2.
```
Edebug 4,5> DEPOSIT my_variable = 15
```

3.
```
Edebug 4,5> DEPOSIT/ASCII:10 %X200 = 'ABCDEF'
```

4.
```
Edebug 4,5> DEPOSIT data_block_item.packet[i] := i
```

EVALUATE

Evaluates an expression.

Format

EVALUATE *expression*

/ADDRESS
/BINARY
/DECIMAL
/HEX
/OCTAL

Parameter

expression
The expression to be evaluated. If you specify the /ADDRESS qualifier, you must specify an address expression.

Qualifiers

/ADDRESS
Displays the effective address of the address expression. This address would be the one examined if the same address expression were specified with the EXAMINE command.

/BINARY
Specifies the display radix as binary if the expression's result is an integer.

/DECIMAL
Specifies the display radix as decimal if the expression's result is an integer.

/HEX
Specifies the display radix as hexadecimal if the expression's result is an integer.

/OCTAL
Specifies the display radix as octal if the expression's result is an integer.

Description

You can evaluate expressions with or without specifying the EVALUATE command. For example, the following expressions are evaluated without specifying the EVALUATE command:

```
Edebug 4,5> (%X1F <> 31) or (%B'11111' <> %037)
FALSE
```

```
Edebug 4,5> b1 <> (b2 and FALSE)
TRUE
```

The following examples use the backslash command (\), which returns the last value:

```
Edebug 4,5> counter+50 * 100
 100600
Edebug 4,5> 2 + \
 100602
Edebug 4,5> 2 + \
 100604
```

Examples

1.
   ```
   Edebug 4,5> EVALUATE 100+200
   300 (0000012C)
   ```

2.
   ```
   Edebug 4,5> EVALUATE/BINARY 100+200
   100101100 (0000012C)
   ```

3.
   ```
   Edebug 4,5> EVALUATE/HEX 100+200
   0000012C
   ```

4.
   ```
   Edebug 4,5> EVALUATE/DECIMAL 100+200
   300 (0000012C)
   ```

5.
```
Edebug 4,5> EVALUATE/OCTAL 100+200
454 (0000012C)
```

6.
```
Edebug 4,5> EVALUATE/ADDRESS VAR_10
7FFFD2D4
```

7.
```
Edebug 4,5> EVALUATE/BINARY/ADDRESS VAR_10
1111111111111111101001011010100 (7FFFD2D4)
```

8.
```
Edebug 4,5> EVALUATE/HEX/ADDRESS VAR_10
7FFFD2D4
```

EXAMINE

Examines the value in a location in the target system's memory.

For an item with a defined data type, the data type determines the amount of storage examined and the format of the display. The qualifiers can override the size associated with the item's type and, in the case of the /ASCII and the floating-point qualifiers, determine the format of the display. The qualifiers also override the display radix specified by the SET MODE command.

NOTE

> When you are examining a pointer, the default display mode for the address is hexadecimal. To override the default, use the /DEC or /OCT qualifiers.

If the command has no associated data type, INTEGER (longword) is the default.

Format

EXAMINE *⟦address⟧*

/ASCII
/BINARY
/BYTE
/DECIMAL
/DOUBLE
/D_FLOAT
/FLOAT
/G_FLOAT
/GRAND
/HEX
/H_FLOAT
/HUGE
/LONGWORD
/OCTAL
/QUADWORD
/REAL
/WORD

Parameter

address
An expression for the address of the value that is to be examined. If the debugger symbol table is present, you can specify a variable reference. This parameter is optional. If you do not specify an address expression, the command examines the next value after the one most recently examined. If you specify a circumflex (^), the command examines the value before the one most recently examined.

Qualifiers

/ASCII
Specifies that the target is a string.

`/ASCII:`*`size`*

Specifies that the target is a string of a given size. The size specifier can be:

:integer
=integer
:(expression)
=(expression)

The default size is 1.

/BYTE

Displays the data in a dump format consisting of a hexadecimal display on the left, followed by an address expression, followed by an ASCII display on the right. The hexadecimal display represents 16 bytes with the lowest address on the right and the highest on the left. The ASCII display has the lowest address on the left and the highest on the right. The address expression indicates lowest address. This qualifier specifies that the target is an arbitrary sequence of bytes.

/BYTE:*size*

Specifies that the target is a byte sequence of a given size. The size specifier can be:

:integer
=integer
:(expression)
=(expression)

The default size is 1.

/BINARY

Specifies that the result is to be displayed as binary if the result is an integer. If the address expression is a variable reference, the variable's data type determines the default display.

/DECIMAL

Specifies that the result is to be displayed as decimal if the result is an integer. If the address expression is a variable reference, the variable's data type determines the default display.

/D_FLOAT

Specifies that the target is a D_floating value.

/DOUBLE

Specifies that the target is a D_floating value.

/FLOAT
Specifies that the target is an F_floating value.

/G_FLOAT
Specifies that the target is a G_floating value.

/GRAND
Specifies that the target is a G_floating value.

/HEX
Specifies that the result is to be displayed as hexadecimal if the result is an integer. If the address expression is a variable reference, the variable's data type determines the default display.

/H_FLOAT
Specifies that the target is an H_floating value.

/HUGE
Specifies that the target is an H_floating value.

/LONGWORD
Specifies that the target is an integer. If the address expression is a variable reference, the variable's data type determines the default display.

/OCTAL
Specifies that the result is to be displayed as octal if the result is an integer. If the address expression is a variable reference, the variable's data type determines the default display.

/QUADWORD
Specifies that the target is a quadword. If the address expression is a variable reference, the variable's data type determines the default display.

/REAL
Specifies that the target is an F_floating value.

/WORD
Specifies that the target is an integer. If the address expression is a variable reference, the variable's data type determines the default display.

Examples

1.
```
Edebug 4,5> EXAMINE R1
R1:     300     (0000012C)
Edebug 4,5> EXAMINE .
R1:     304     (00000130)
```

2.
```
Edebug 4,5> EXAMINE %X4000
4000:  300     (0000012C)
Edebug 4,5> EXAMINE . + %X10
4010:  34      (00000130)
```

3.
```
Edebug 4,5> EXAMINE R0
R0:     10      (0000000A)
Edebug 4,5> EXAMINE .
R0:     10      (0000000A)
```

4.
```
Edebug 4,5> EXAMINE
R1:     20      (00000014)
Edebug 4,5> EXAMINE ^
R0:     10      (0000000A)
```

5.
```
Edebug 4,5> EXAMINE/ASCII S1
S1: A
```

6.
```
Edebug 4,5> EXAMINE/ASCII:10 S1
S1: ABC
```

7.
```
Edebug 4,5> EXAMINE/BY S1
S1:  97 (00000061)
```

8.
```
Edebug 4,5> EXAMINE/BY:1 S1
                                        61|  S1  | a
```

9.
```
Edebug 4,5> EXAMINE/BY:3 S1
                              636261|  S1  | abc
```

10.
```
Edebug 4,5> EXAMINE/BY:10 S1
               2020 20202020 20636261|  S1  | abc
```

11.
```
Edebug 4,5> EXAMINE/BY:20 S1
     00064C50 00002020 20202020 20636261|  S1  | abc          ..PL..
                                00064CD0|  S1 + 0010  |  .L..
```

12.
```
Edebug 4,5> EXAMINE/WORD S1
S1:  25185 (00006261)
```

13.
```
Edebug 4,5> EXAMINE/QUAD S1
                   20202020 20636261 |  S1  | abc
```

14.
```
Edebug 4,5> EXAMINE/LONG S1
S1:  543384161 (20636261)
```

15.
```
Edebug 4,5> EXAMINE/DOUBLE R0
R0:      1.00000000000000000E+00005
```

16.
```
Edebug 4,5> EXAMINE/G_FLOAT R0
R0:      8.41178011028559882E+00041
```

17.
```
Edebug 4,5> EXAMINE/HUGE R0
R0:      1.06499995818682317E+00675
```

18.
```
Edebug 4,5> EXAMINE/OCTAL I1
I1:  1 (00000001)
```

19.
```
Edebug 4,5> EXAMINE/BINARY I1
I1:  1 (00000001)
```

20.
```
Edebug 4,5> EXAMINE/HEX I1
I1: 00000001
```

21.
```
Edebug 4,5> EXAMINE/BY:3/OCTAL I1
00000001 | I1 |  . . .
```

22.
```
Edebug 4,5> EXAMINE/BY:4/BINARY I1
              1   | I1 |  . . .
```

23.
```
Edebug 4,5> EXAMINE data_block_item
next_data_block:  7FFFD3AC
PACKET(1):  33 (00000021)
PACKET(2):  2 (00000002)
PACKET(3):  0 (00000000)
PACKET(4):  0 (00000000)
BIT_ITEM: True
```

24.
```
Edebug 4,5> EXAMINE arr_item_10
arr_item_10(1):  33 (00000021)
arr_item_10(2):  2 (00000002)
arr_item_10(3):  0 (00000000)
arr_item_10(4):  0 (00000000)
arr_item_10(5):  0 (00000000)
arr_item_10(6):  0 (00000000)
arr_item_10(7):  0 (00000000)
arr_item_10(8):  0 (00000000)
arr_item_10(9):  0 (00000000)
arr_item_10(10):  805306368 (30000000)
```

EXAMINE/INSTRUCTION

Displays a machine-instruction sequence.

Format

EXAMINE/INSTRUCTION *[[start]] [[:end]]*

Parameters

start
An expression for the address at which the command is to start displaying the machine-instruction sequence. This parameter is optional. If you omit this parameter and the *end-address* parameter, the command displays the instruction at the location after the last-examined location.

end
An expression for the address at which the command is to stop displaying the machine-instruction sequence. This parameter is optional. If you omit it, the command displays one instruction.

Examples

1.
```
Edebug 4,5> EXAMINE/INSTRUCTION %X4000
4000: MOVAB    -10(fp),-14(fp)
Edebug 4,5> SET BREAK .
```

2.
```
Edebug 4,5> EXAMINE/INSTR %LABEL first_label :
More?> %LABEL first_label + 40
%Line 345 + 0000: PUSHL  #0B
%Line 345 + 0002: PUSHL  #00
%Line 345 + 0004: PUSHAB $CODE + 00AE
%Line 345 + 0008: PUSHL  #0B
%Line 345 + 000A: PUSHAB PAS$OUTPUT\$DATA + 0010
%Line 345 + 0010: CALLS  #05,@0000166C
%Line 345 + 0017: PUSHAB PAS$OUTPUT\$DATA + 0010
%Line 345 + 001D: CALLS  #01,@00001664
Edebug 4,5> EXAMINE/i
%Line 346 + 0000: MOVC3  #+ 0093,-16E8(FP)
```

EXAMINE/PSL

Displays the value at a specified location and expands the value into an ASCII display of a processor status longword (PSL).

Format

EXAMINE/PSL *[[address]]*

Parameter

address
An expression for the address of the value to be displayed and expanded. The parameter is optional. If you omit it, the value at the current address is displayed and expanded.

Examples

1.
```
Edebug 4,5> EXAMINE/PSL @SP
```

2.
```
Edebug 4,5> EXAMINE/PSL PSL
CM  TP  FPD  IS  Current Mode  Previous Mode  IPL  DV  FU  IV  T  N  Z  V  C

--  --  ---  --  ------------  -------------  ---  --  --  --  -  -  -  -  -
0   0    0   0              3              3    0   1   0   1  0  0  0  0  0
```

EXAMINE/SOURCE

Displays the source-line sequence.

You can use this command only with the remote debugger.

Format

EXAMINE/SOURCE *[[start]] [[:end]]*

Parameters

start
An expression for the address at which the command is to start displaying the source-line sequence. This parameter is optional. If you omit this parameter and the *end-address* parameter, the command displays the source line at the location after the last-examined location.

end
An expression for the address at which the command is to stop displaying the source-line sequence. This parameter is optional. If you omit it, the command displays one source line.

Examples

1.
```
Edebug 4,5> EXAMINE/SOURCE %LINE 345
Module  TSTEDEBUG
345:  WRITELN('first label');
```

2.
```
Edebug 4,5> EXAMINE/SOURCE %LABEL first_label:
More?> %LABEL first_label + 40
Module  TSTEDEBUG
345:  WRITELN('first label');
346:
347: call_successor_label:
348:
349: p_vs1 := 'this is the main body value';
```

3.
```
Edebug 4,5> EXAMINE/SOURCE
Module  TSTEDEBUG
350:  SUCCESSOR;
```

EXIT

Exits the debugging session.

When you use the remote debugger, the EXIT command disconnects the debugger from the system but sets up a way for you to connect later with a process under debugger control in the same state as when you disconnected. For example, breakpoints are preserved.

Typing CTRL/Z is equivalent to using the EXIT command.

Format

EXIT

GO

Continues a session's execution.

Format

GO *⟦address⟧*

Parameter

address
An expression for the address at which the command is to continue execution. This parameter is optional. If you specify it, the integrity of the program state is not guaranteed. If you do not specify it, the session continues execution from where it stopped.

HALT

Stops a process by raising an asynchronous exception. If the process being stopped is under the debugger's control or does not handle the exception, the process enters the debugger's command wait state. Otherwise, the process might abort. The debugger reports an exception for the process if the process is not under debugger control.

Format

HALT *⟦process⟧ ⟦node⟧*

Parameters

process
The process identifier of the process to be stopped. This parameter is optional. If you omit it, the command stops the current process.

node
The node on which the process to be stopped resides. This parameter is optional. If you omit it, the command stops a process on the current node. Use this parameter if you are debugging multiple nodes.

Description

The debugger performs the HALT operation with a VAXELN kernel service that signals the target process. Once the signal takes effect, the debugger gains control, and the process enters the debugger's command wait state.

NOTE

You cannot halt a process when it is in one of the following states:

- The process cannot be scheduled to run because of the scheduling state of the system; to be halted, the process must be executing or ready to execute.
- The process is waiting for the execution of the ACCEPT_CIRCUIT or CONNECT_CIRCUIT kernel service to complete.

- The process is in an implicit wait state during the execution of the SEND kernel service to a circuit that is full.
- The process is a kernel-mode process running at an elevated interrupt priority level (IPL).

In each of these cases, the debugger gains control when the process leaves the state in which it cannot be halted.

HELP

Displays information on each of the debugger commands.

When you are using the EDEBUG utility, the HELP items are the same as those displayed by the VAX/VMS HELP command.

Format

HELP *[[keyword]]*

Parameters

keyword
A keyword that refers to the topic or subtopic on which you want information. Information within HELP is arranged in a hierarchical manner. The levels are:

1. None — If you do not specify a keyword, HELP lists the topics that are documented in the help file. Each item in the list is a keyword in the first level of the hierarchy.
2. Topic — If you specify a keyword that names a topic, HELP describes the topic as it is documented in the help file. Keywords for additional information available on this topic are listed.
3. Topic subtopic — If you specify a subtopic following a topic, HELP provides a description of the specified subtopic.

Description

To use the HELP facility in its simplest form, issue the HELP command from your terminal. HELP displays a list of topics at your terminal. If you want to see more information on one of the topics, type HELP *topic* at the command level prompt. The system will display information on that topic. If the topic has subtopics, HELP will list the subtopics. If you want information on one of the subtopics, type HELP *topic subtopic* at the command level prompt.

IF

Conditionally executes a 1-line command based on the value of a Boolean expression. (You can use commands created with SET COMMAND.)

Format

IF *boolean-expression* **THEN** *one-line-command*

Parameters

boolean-expression
A Boolean expression whose value the IF command uses to determine whether to execute a specified command.

one-line-command
The 1-line command that is to be executed based on the evaluation of the specified Boolean expression.

Examples

1.
```
Edebug 4,5> IF a = (123+RO) THEN DEPOSIT c := 25
```

2.
```
Edebug 4,5> IF a = (123+RO) THEN EVALUATE dump_symbols
```

LEAVE

Aborts the execution of a nested command file and returns to the next higher level. The LEAVE command is useful in command files as an argument in an IF command.

Format

LEAVE

Example

```
Edebug 4,5> IF a = (123+R0) THEN LEAVE
```

LOAD

Installs a new program image into the target system. You can then execute the new program image by using the CREATE JOB command. The LOAD command is a simplified interface to the program-loading facilities and is not available if you are using the local debugger.

Format

LOAD *program file*

/DEBUG
/JOB_PRIORITY=*n*
/KERNEL⟦=*n*⟧

Parameters

program
The name of the program image to be installed on the target system. The name you specify must be unique. This name can subsequently be used in a CREATE JOB command.

file
The name of the file for the program to be installed on the target system. You must include the file type, for example, .EXE. The program image file is opened in the context of the target system, not in the context of the remote debugger. Therefore, to load a program image from another node, you must include the node in the file specification. Node names are not valid; you must specify the node in the following format:

area.node-number

For example:

5.014

Qualifiers

/DEBUG

Specifies that the remote debugger is to get control after the specified program is started on the target system using the CREATE JOB command. If you do not specify this qualifier, the job running the program will not be under debugger control.

/JOB_PRIORITY=n

Indicates the priority at which the job running the specified program is to execute. The default is 16. The process priority always defaults to 8.

/KERNEL⟦=n⟧

Indicates that the specified program is to run in kernel mode. The parameter *n* specifies the size of the kernel's stack. The default is 1.

PREDECESSOR

Moves the session's reference scope back a specified number of call frames in the calling order.

For example, PREDECESSOR 1 lets you use variable names or other names declared in the routine that called the current routine. The context in which the session is stopped is not affected.

Format

PREDECESSOR *⟦expression⟧*

/AUTO

Parameter

expression
An expression whose value indicates the number of call frames the reference scope is to be moved. This parameter is optional. If you omit it, the reference scope moves back one call frame.

Qualifier

/AUTO
Finds the most recent frame described by the program traceback information when a process stops.

SEARCH

Searches the program for a string or identifier. If no match is found, the command prompt returns. No message is displayed.

Format

SEARCH *⟦range⟧ ⟦target⟧*

/ALL
/NEXT

Parameters

range
The range of the search, which you can specify one of the following ways:

- *module* — Search the named module beginning at its first line and continuing to its end.
- *module\line* — Begin the search at the specified line number in the named module.
- *module\line:line* — Search the specified range of line numbers in the specified module.
- *line* — Begin the search at the given line number in the current module.
- *line:line* — Search the specified range in the current module.

On the first SEARCH command, you must specify a *range*. If you omit this parameter in subsequent SEARCH commands, the debugger searches the module most recently searched, beginning at the first line after the one most recently searched and continuing to the end of that module.

target
The string or identifier for which the debugger is to search. The debugger displays the source lines containing occurrences of the value specified.

SEARCH

On the first SEARCH command, you must specify a *target*. If you omit this parameter in subsequent search commands, the debugger searches for the *target* most recently specified. If you specify a *target*, you must also specify a *range*.

Qualifiers

/ALL

Displays all occurrences of the specified string or identifier in the specified range.

/NEXT

Displays the first occurrence of the specified string or identifier in the specified range. This qualifier is the default.

Examples

1.
```
Edebug 4,5> SEARCH TSTEDEBUG\ 1 : 100 'A'
Module  TSTEDEBUG
5: PROCEDURE macsub; SEPARATE;
```

2.
```
Edebug 4,5> SEARCH 1 : 100 VAR
Module  TSTEDEBUG
6: FUNCTION foraddf(VAR ii,jj : INTEGER);
```

3.
```
Edebug 4,5> SEARCH
Module  TSTEDEBUG
7: PROCEDURE foradds(VAR ii,jj : INTEGER);
```

4.
```
Edebug 4,5> SEARCH/ALL
Module TSTEDEBUG
20:    VAR in_out_str : STRING(<k>)
100:   VAR param_string : STRING(<k>)
```

SET BREAK

Sets a breakpoint at a specified address.

NOTE

When a debug symbol table is present and you use a VAX procedure name to set a breakpoint at the procedure's entry point, the debugger places the breakpoint at that procedure's first instruction. For example:

```
SET BREAK writeroutine
```

This command automatically sets the breakpoint at the address of *writeroutine*+2, the first instruction after the entry mask. If no debug symbol table is present, you must place the breakpoint at the procedure's first instruction yourself.

Format

SET BREAK *⟦address⟧* **DO** *⟦command⟧*

/ALL
/JOB
/KERNEL

Parameters

address
An expression for the address where the breakpoint is to be set.

command
A command to be executed when the specified breakpoint is reached. Only a 1-line command is allowed. This parameter is optional. You can specify commands defined by SET COMMAND.

Qualifiers

/ALL
Specifies that the breakpoint is valid for the entire job. This qualifier is equivalent to /JOB. If you do not specify the /ALL or /JOB qualifier, the breakpoint affects only the current session's process.

/JOB
Specifies that the breakpoint is valid for the entire job. This qualifier is equivalent to /ALL. If you do not specify the /ALL or /JOB qualifier, the breakpoint affects only the current session's process.

/KERNEL
Sets a breakpoint in the kernel's breakpoint data base. You can use this qualifier only with kernel-mode programs, and you cannot specify a command. The breakpoint is set in the process's memory, and the S0 kernel address of the breakpoint is recorded in the kernel's breakpoint data base.

You should use /KERNEL to set breakpoints in interrupt service routines and other kernel mode code that is executed at an elevated IPL. These jobs must be debugged in the kernel debugging session. Because the S0 address of the breakpoint is set in the kernel's breakpoint data base, you can set breakpoints in interrupt service routines by referring to their program address space. The breakpoint in the interrupt service routine occurs when your program is executing out of S0 space in the kernel, not necessarily in the context of the program itself (see the SET SESSION command).

NOTE

You cannot use this qualifier with the remote debugger, unless both the remote and local debuggers were built into the system. In that case, when the breakpoint occurs, the debug information will be displayed at the console terminal. You must then enter commands to the kernel session at the console terminal.

Examples

1.
```
Edebug 4,5> SET BREAK 500+512
```

2.
```
Edebug 4,5> SET BREAK sym_tbl\ %LINE 125
```

3.
```
Edebug 4,5> SET BREAK sym_tbl\ %LABEL not_found
```

4.
```
Edebug 4,5> SET BREAK sym_tbl\srch_rtn\srch_struct
```

5.
```
Edebug 4,5> SET BREAK %LINE 25
```

6.
```
Edebug 4,5> EXAMINE/INSTRUCTION %X4000
4000: MOVAB    -10(fp),-14(fp)
Edebug 4,5> SET BREAK .
```

7.
```
Edebug 4,5> CANCEL BREAK symbol_not_found
```

8.
```
Edebug 4,5> CANCEL BREAK %LINE 25
```

SET COMMAND

Creates a command for use during a session.

Format

SET COMMAND *identifier* **DO** *⟦one-line-command⟧*

Parameters

identifier
An identifier whose occurrence indicates that a command is to execute.

one-line-command
A 1-line command to be substituted for each occurrence of the specified identifier in a command context. This parameter is optional. If you omit it, the debugger prompts you for a sequence of debugger commands, which you terminate by typing an empty line. When the command identifier appears in a command context, the sequence of commands executes.

Examples

1.
```
Edebug 4,5> SET COMMAND R0plus10 (EVALUATE DO R0+10)

Edebug 4,5> SET COMMAND R1plus10
Command> EVALUATE R1+10
Command>
Edebug 4,5> R0plus10
100    (00000064)
Edebug 4,5> SHOW COMMAND R1plus10
R1plus10 - EVALUATE R1+10
```

2.
```
Edebug 4,5> SET COMMAND R0plus10 DO()    ! Erase R0plus10 command
```

SET CONTROL

Reverses the action of the CANCEL CONTROL command; places all processes in the current job under debugger control.

Format

SET CONTROL

SET EXCEPTION BREAK

Causes the session in the debugger to gain control when an exception occurs. The debugger gains control before the kernel searches for a programmed exception handler. By default, the debugger gets control only if no programmed exception handlers are found. When the session stops at an exception break, use the GO command to continue the search for an exception handler.

You can cancel the state created by the SET EXCEPTION BREAK command by using the CANCEL EXCEPTION BREAK command.

Format

SET EXCEPTION BREAK

SET LOG

Opens the specified file and starts logging the session. You can turn logging off and on again without closing the file by using the SET OUTPUT NOLOG and SET OUTPUT LOG commands. The file is closed when you exit or when you enter a SET LOG command without a file specification.

Commands and responses are logged. Responses are preceded with an exclamation point (!) making them comment lines. You can later submit the log file as a debugger command file.

You can use this command only with the remote debugger.

Format

SET LOG *filespec*

Parameter

filespec
The file specification of the file to which the logging information is to be written. The default file type is .LOG.

SET MODE

Changes debugger command modes. You can change multiple modes in one SET MODE command. Use the SHOW MODE command to see the state of the debugger modes that are set.

Format

SET MODE *keyword ⟦, keyword,... ⟧*

Parameter

keyword

A keyword indicating the type of mode that is to be set. Table 5–3 lists the valid keywords with brief descriptions.

Table 5–3: SET MODE Keywords

Keyword	Description
	Integer Display Mode
DECIMAL	Sets the default display radix to decimal.
HEXADECIMAL	Sets the default display radix to hexadecimal.
OCTAL	Sets the default display radix to octal.
	Floating Point Mode
D_FLOAT	Sets the default floating-point precision to the D_floating data type. This data type ranges from .29e–38 to 1.7e38.
DOUBLE	Sets the default floating-point precision to the D_floating data type. This data type ranges from .29e–38 to 1.7e38. DOUBLE is the default.
G_FLOAT	Sets the default floating-point precision to the G_floating data type. This data type ranges from .56e–308 to .9e308. The G_floating data type is not present on some VAX models.
GRAND	Sets the default floating-point precision to the G_floating data type. This data type ranges from .56e–308 to .9e308. The G_floating data type is not present on some VAX models.

Table 5–3 (Cont.): SET MODE Keywords

Keyword	Description
	Step Unit Mode
INSTRUCTION	Sets the default step unit to an instruction. You can also change the step mode by using the SET STEP command.
LINE SOURCE	Sets the default step unit to a program source line. This mode is only valid in the remote debugger and only if a program's symbol table is available. If a session stops in a section of the program written in a higher-level language, the debugger is usually set in line mode. You can also change the step mode by using the SET STEP command. LINE and SOURCE are equivalent.
	Into/Over Mode
INTO	Specifies that you want to step into a subroutine to inspect its code.
OVER	Specifies that you want to step over a subroutine.
	Verify Mode
NOVERIFY	Suppresses the display of substituted commands and commands from command files as they execute.
VERIFY	Displays substituted commands and commands from command files as they execute.
	Prompt Mode
NOPROMPT	Prevents the debugger from displaying a prompt until you type a CTRL/C. Because debuggers use the same terminal as the console I/O, you might use this keyword to free your terminal for use by another program.

NOTE

When you are examining a pointer, the default display mode for the address is hexadecimal, regardless of the mode set with the SET MODE command. To override this default, use the /DEC or /OCT qualifiers with the EXAMINE command.

SET MODE

Examples

1.
```
Edebug 4,5> SET MODE OCTAL
Edebug 4,5> 1234
1234 (0000029C)
```

2.
```
Edebug 4,5> SET MODE HEX
Edebug 4,5> 1234
00001234
```

3.
```
Edebug 4,5> SHOW MODE
Radix is decimal.
Floating conversion mode is double.
Step over routine calls by line.
Automatic commands are not verified.
```

SET OUTPUT

Lets you turn logging on or off without closing the log file.

Format **SET OUTPUT** *[[NO]]LOG*

SET PROGRAM

Informs the debugger that the session's job is running a specified copy of a program image. This command is necessary when the debugger fails to access a program's image file, because you are using a new copy of the image file or because the file's protection prevents access. For example, when you copy an .EXE file, its version number might not be the same as the number recorded in the .SYS. Likewise, the directory of a copy of an .EXE file might not be the same as the directory recorded in the .SYS file. The SET PROGRAM overrides the file specification in the .SYS file.

The debugger copies the specified file's symbol table information into its memory. The debugging mode changes automatically, based on the module in which the session stops. The radix is set appropriately for the source language.

This command affects sessions associated with the current job.

Format

SET PROGRAM *⟦image-file-spec⟧*

Parameter

image-file-spec
The program image to be run by the session's job. If you specify the name of a file that does not exist, the debugger displays an error message. This parameter is optional. However, if you omit it, the debugger discards the symbol table information for the current image.

SET RETURN BREAK

Stops the session when the current routine returns. This command enables a temporary breakpoint that is encountered when the routine is about to return.

You cannot cancel the state created by the SET RETURN BREAK command by using the CANCEL BREAK command.

Format

SET RETURN BREAK

SET SESSION

Changes the session to a debugging session on the current node or on another node already specified with the DEBUG command. The state of the previous session remains the same.

Format

SET SESSION *process* *[[node]]*

/GO
/KERNEL

Parameters

process
The process identifier for the debugging session you want. Refer to Section 5.3.1 for a discussion of process identifiers.

node
The node name of the process whose debugging session you want. This parameter is optional. If you omit it, the command changes the debugging session on the current node. You must use this parameter to change nodes, if you are debugging multiple nodes.

Qualifiers

/GO
Passes the GO command to the current session, then changes the session.

/KERNEL
Causes the kernel debugging session to take control at the target console. This qualifier is valid only in the local debugger.

Examples

1. `Edebug 4,5> SET SESSION 5,2`

2. `Edebug 4,5> SET SESSION 5,2 NODE2`

3. `Edebug 4,5> SET SESSION MYPROG`

SET SOURCE

Specifies a source file that is to be used during the session. Use this command when you build an application on one machine and run the application on another machine, when you move the source file to a new directory, or when you want to use another version of the source file.

Format

SET SOURCE *filespec*

Parameter

filespec
A complete VAX/VMS file specification; for example, DUA0:[SRC]PRG.PAS.

SET STEP

Changes the default action of steps with respect to procedures and functions.

See the SET MODE command description for more information on step actions.

Format

SET STEP *⟦step-type⟧ ⟦step-size⟧*

Parameters

step-type
The keyword INTO or OVER, which indicates whether the command is to step into or over a routine. The default is OVER.

step-size
The keyword INSTRUCTION, LINE, or SOURCE, which indicates the size of the step the command is to take. (SOURCE and LINE have the same meaning.) The default is LINE.

SET TIME

Sets the system time on a specified node.

Format

SET TIME *time-string node*

Parameters

time-string
A time string specified in the standard format for absolute times (*dd-mmm-yyyy hh:mm:ss.cc*).

node
The name of the node on which the system time is to be set. The current node is the default.

SHOW BREAK

Displays information about all breakpoints.

Format

SHOW BREAK

Example

```
Edebug 4,5> SHOW BREAK
Module name      Routine or Psect name      Line      Rel PC      Abs PC
TSTEDEBUG        TSTEDEBUG                  345       000002F1    00000FBF
TSTEDEBUG        TSTEDEBUG                  353       00000327    00000FF5
TSTEDEBUG        TSTEDEBUG                  356       0000034B    00001019
TSTEDEBUG        TSTEDEBUG                  359       0000036F    0000103D
```

SHOW CALLS

Displays the call history for the reference scope.

Format

SHOW CALLS

Example

```
Edebug 4,5> SHOW CALLS
Module name     Routine or Psect name     Line    Rel PC      Abs PC

TSTEDEBUG       TSTEDEBUG                 502     00000202    00000ED0

TSTEDEBUG       TSTEDEBUG                 217     00000002    00000CD0
                                                  00000000    800024F5
```

SHOW COMMAND

Displays commands defined by SET COMMAND.

Format

SHOW COMMAND *⟦identifier⟧*

/ALL

Parameter

identifier
A string that identifies a command previously set with SET COMMAND. This parameter is optional. If you omit it, specify the /ALL qualifier.

Qualifier

/ALL
Displays all commands previously set with SET COMMAND.

Example

See the SET COMMAND command description for an example of SHOW COMMAND.

SHOW JOB

Displays information about the processes in a job.

Format

SHOW JOB ⟦*job*⟧ ⟦*node*⟧

/ALL

Parameters

job
A string, identifier, or integer indicating the job for which information is to be displayed. This parameter is optional. If you omit it, the debugger displays information about the current job.

node
The name of the node on which the job resides. This parameter is optional. If you omit it, the debugger displays the processes in the current or specified job on the current node. Use this parameter if you are debugging multiple nodes.

Qualifier

/ALL
Displays all jobs.

Examples

1.
```
Edebug 4,5> SHOW JOB 6
Job 6, program testload, priority 16 is waiting.
  Shared read/write size: 1024.
   Process 1, priority 8, in debug command wait.
     Stack size: 5632. CPU time:   0 00:00:00.02
  Accumulated CPU time for this job:   0 00:00:00.02
```

2.
```
Edebug 4,5> SHOW JOB testload
 Job 6, program testload, priority 16 is waiting.
  Shared read/write size: 1024.
   Process 1, priority 8, in debug command wait.
     Stack size: 5632. CPU time:   0 00:00:00.02
   Accumulated CPU time for this job:   0 00:00:00.02
```

SHOW MESSAGE

Displays the message text associated with an expression value's return status.

Format

SHOW MESSAGE *expression*

Parameter

expression
The return status of an expression value for which information is to be displayed.

Example

```
Edebug 4,5> SHOW MESSAGE %X7C3C
Bad parameter value
```

SHOW MODE

Displays the debugger's current operating modes (stepping defaults, radix, and floating-point conversion type).

See the SET MODE command description for a SHOW MODE example.

Format **SHOW MODE**

SHOW MODULE

Displays information about the program associated with the session.

Format

SHOW MODULE

Example

```
Edebug 4,5> SHOW MODULE
Program- SEA$:[DEBUG.TEST]TSTEDEBUG.EXE;62
Module name     Symbols     Language     Source
TSTEDEBUG       Yes         Pascal       Yes
FORADDF         Yes         FORTRAN      No
FORADDS         Yes         FORTRAN      No
PASSUBS         Yes         Pascal       Yes
MAINO           No          Macro        No
PLISUB          Yes         PL/I         Yes
BINARY          Some        Bliss        Yes
PAS$INPUT       No          Macro        No
PAS$OUTPUT      No          Macro        No
```

SHOW PROCESS

Displays the system state of a job or a process in a job.

Format

SHOW PROCESS *[[process]] [[node]]*

/ALL

Parameters

process
The name of the process for which the system state is to be displayed. This parameter is optional. If you omit it and do not specify /ALL, the debugger displays the state of the process associated with the current debugging session.

node
The name of the node on which the process or job resides. This parameter is optional. If you omit it, the debugger displays the state of the processes on the current node. Use this parameter if you are debugging multiple nodes.

Qualifier

/ALL
Displays all jobs.

Examples

1.
```
SHOW PROCESS
Job 6, program TESTLOAD, priority 16 is waiting.
 Shared read/write size: 1024.
  Process 1, priority 8, in debug command wait.
    Stack size: 5632. CPU time:   0 00:00:00.02
```

2.
```
Edebug 4,5> SHOW PROCESS 2,4
Job 2, program XQDRIVER, priority 1 is waiting.
  Shared read/write size: 30720.
   Process 4, priority 8, is waiting.
     Stack size: 2048. CPU time:   0 00:00:00.07
```

SHOW PROGRAM

Displays the system information about an installed program.

Format

SHOW PROGRAM *[[program]]*

/ALL

Parameter

program
The name of the program for which system information is to be displayed. This parameter is optional. If you omit it, specify the /ALL qualifier.

Qualifier

/ALL
Displays system information about all the installed programs.

Example

```
Edebug 4,5> SH PROGRAM TESTLOAD
Program: TESTLOAD   Debug  User mode
  Default job priority: 16   Default process priority: 8
  Kernel stack size: 8       User stack size: 2
  Filename: "SEA$:[DEBUG.TEST]TESTLOAD.EXE;75"
```

SHOW SESSION

Displays the debug state of debugging sessions.

Format

SHOW SESSION *⟦process⟧ ⟦node⟧*

/ALL

Parameters

process
The name of the process whose debug state is to be displayed. This parameter is optional. If you omit it, the debugger displays the state of the current session.

node
The name of the node on which the process resides. This parameter is optional. If you omit it, the debugger displays the state on the current node. Use this parameter if you are debugging multiple nodes.

Qualifier

/ALL
Displays the debug state for all sessions in the current job.

Example

```
Edebug 4,5> SHOW SESSION 5,1
Job 5, process 1, program TSTEDEBUG needs attention.
Module  TSTEDEBUG
216:
>>217: BEGIN
  218:  gbldef_i := 4;
  219:  gbldef_j := 12345;
```

SHOW SYMBOL

Displays the debugger's symbol table information, providing a way to see what the debugger knows about the symbols in a reference scope.

Format

SHOW SYMBOL *⟦pathname⟧ ⟦identifier⟧*

/DEFINE

Parameters

pathname
The name of the path identifying the reference scope for which symbols' symbol table information is to be displayed. This parameter is optional. If you omit it, the debugger displays symbol table information for symbols in the current reference scope. If you specify a path name and an identifier, the debugger displays information for the specified symbol in the specified scope.

identifier
The symbol for which symbol table information is to be displayed. This parameter is optional. If you omit it, the debugger displays information for all symbols. If you specify a path name and an identifier, the debugger displays information for that symbol in the specified scope. If you specify an identifier without a path name, the debugger displays the symbol from the current scope.

Qualifier

/DEFINE
Displays symbol table information about a debugger-defined symbol. You can specify this qualifier with the *identifier* parameter. If you do not specify an identifier, the debugger displays all defined session variables.

Examples

1.
```
Edebug 4,5> SHOW SYMBOL
Outer Scope:
1 gbldef_i
    Type:  Integer
    Size: 1 Longword
    Located at address 00000204

1 gbldef_j
    Type:  Integer
    Size: 1 Longword
    Located at address 00000200

Routine: TSTEDEBUG
1 APTR
    Type:  Pointer to anytype
    Size: 1 Longword
    Located at -0000175C(FP)

1 arr_item_10
    Type:  Array of Integer
    Size:  Not determined at compile time
    Located at -000015D8(FP)
```

2.
```
Edebug 4,5> SHOW SYMBOL gbldef_i
Outer Scope:
1 gbldef_i
    Type:  Integer
    Size: 1 Longword
    Located at address 00000204
```

3.
```
Edebug 4,5> SHOW SYMBOL/DEFINE
$:  Relocation
NIL:  Relocation
iii:  Integer
ptr1: pointer to  Integer
s1:  String(10)
relo:  Relocation
```

SHOW SYSTEM

Displays the memory, CPU time, and jobs of the system. The display of available memory refers to the total number of pages of memory available; these pages are not necessarily contiguous.

Format

SHOW SYSTEM *[[node]]*

Parameter

node

The name of the node for which system information is to be displayed. This parameter is optional. If you omit it, the debugger displays information about the current system. Use this parameter if you are debugging multiple nodes.

Example

```
Edebug 4,5> SHOW SYSTEM
Available: Pages: 1335, Page table slots: 47, Pool blocks: 216
Time since SET TIME: Idle:  0 00:00:23.10 Total:   0 00:00:23.73
Time used by past jobs:   0 00:00:00.03

Job 2, program XQDRIVER, priority 1 is waiting.
Job 3, program EDEBUGREM, priority 3 is running.
Job 4, program DUDRIVER, priority 4 is waiting.
Job 5, program FALSERVER, priority 16 is waiting.
Job 6, program TESTLOAD, priority 16 is waiting.
```

SHOW TIME

Displays the time.

Format

SHOW TIME *⟦node⟧*

Parameter

node

The name of the node on which the time is to be displayed. This parameter is optional. If you omit it, the debugger displays the time on the current system.

SHOW TRANSLATION

Displays the translation (PORT object value) associated with a specified name.

Format

SHOW TRANSLATION *name ⟦node⟧*

Parameters

name
An identifier or quoted string for which a translation is to be displayed. Use a quoted string to allow characters that are not valid in identifiers.

node
The name of the node on which the translation is to be displayed. This parameter is optional. If you omit it, the debugger displays the translation on the node associated with the current session.

Examples

1.
```
Edebug 4,5> SHOW TRANSLATION CONSOLE
Node: AA-00-04-00-E0-20, Network: 0, Object: 131316
000020E0 000400AA 00000000 000200F4 | 00000000 | .............. ..
```

2.
```
Edebug 4,5> SHOW TRANSLATION 'CONSOLE'
Node: AA-00-04-00-E0-20, Network: 0, Object: 131316
000020E0 000400AA 00000000 000200F4 | 00000000 | .............. ..
```

3.
```
Edebug 4,5> SHOW TRANSLATION CONSOLE SEA4
Node: AA-00-04-00-E0-20, Network: 0, Object: 131316
000020E0 000400AA 00000000 000200F4 | 00000000 | .............. ..
```

STEP

Executes the next instruction or line. This command executes an instruction or line and returns control to the user.

See the SET MODE command description for more information on step actions. The STEP command qualifiers temporarily override any conflicting mode as set by default or by the latest SET MODE or SET STEP command.

Format

STEP *⟦expression⟧*

/INSTRUCTION
/INTO
/LINE
/OVER
/SOURCE

Parameter

expression
An expression that indicates the number of times the command is to repeat. This parameter is optional. If you omit it, the debugger steps one line or instruction.

Qualifiers

/INSTRUCTION
Explicitly steps over one instruction.

/INTO
Stops in a routine if the next instruction or line calls a procedure or function.

/LINE

Steps over the instructions associated with the current line if the associated program has line-number information. If no associated program exists or if the session is not at a point identified by a line number, the debugger steps over one instruction. The /LINE and /OVER qualifiers are the defaults for the STEP command.

/OVER

Stops after a routine's return if the next instruction or line calls a procedure or function. The /LINE and /OVER qualifiers are the defaults for the STEP command.

/SOURCE

Steps over the instructions associated with the current line if the associated program has line-number information. If no associated program exists or if the session is not at a point identified by a line number, the debugger steps over one instruction.

SUCCESSOR

Moves the session's reference scope a specified number of call frames forward in the calling order (following use of the PREDECESSOR command).

Format

SUCCESSOR *[[expression]]*

Parameter

expression
An expression indicating the number of call frames forward that the reference scope is to move. This parameter is optional. If you omit it, the debugger moves forward one call frame. For example, SUCCESSOR 1 lets you use variable names or other names declared in the next routine called from the current routine. The context in which the session is stopped is not affected.

TYPE

Displays the source program lines in a specified range.

Format

TYPE *range*

Parameter

range
The range of source lines to be displayed. Specify the range in the following format:

⟦*module*\⟧ ⟦*expression*⟧ ⟦*:expression*⟧

The expressions must represent line numbers indicating the beginning and end of the range. If you omit the name of the module and the backslash, the debugger assumes the module where the session's reference scope is set or the last module used in a TYPE or SEARCH command. If you omit the line-number expressions, the debugger displays the line after the last TYPE or SEARCH command.

Examples

1.
```
Edebug 4,5> TYPE 1
Module  TSTEDEBUG
1: MODULE TSTEDEBUG;
```

2.
```
Edebug 4,5> TYPE 1 : 5
Module  TSTEDEBUG
1: MODULE TSTEDEBUG;
2:
3: VAR    gbldef_i,gbldef_j : INTEGER;
4:
5: PROCEDURE macsub; SEPARATE;
```

3.
```
Edebug 4,5> TYPE passubs\ 1 : 6
Module  passubs
1: MODULE passubs;
2: PROCEDURE passubs(i,j : INTEGER; k : INTEGER);
3:
4: BEGIN
5:   k := i + j;
6:   END;
```

UNLOAD

Removes a previously loaded program image from the system. This command is not available in the local debugger.

Format

UNLOAD *program ⟦node⟧*

Parameters

program
The name of the program image to be removed from the system.

node
The node from which the program image is to be removed. This parameter is optional. If you omit it, the debugger removes the program image from the current node.

WAIT

Suspends the operation of the remote debugger for a specified time interval. This command is useful in command files after a command that must be completed before the next command can execute. For example, if you attempt to delete a process after halting it, an error occurs if the HALT command has not finished executing.

Format

WAIT *time-specifier*

Parameter

time-specifier
An integer indicating the number of tenths of a second to wait.

Example

```
HALT 7,1
WAIT 5
DELETE PROCESS 7,1
```

Chapter 6

Performance Analysis

The VAXELN Performance Utility is a software development tool that helps you analyze the run-time behavior of VAXELN applications. The VAXELN Performance Utility can pinpoint execution bottlenecks and other performance problems in applications. Using this information, you can modify your VAXELN applications to run faster.

This chapter explains the features of the VAXELN Performance Utility (Section 6.1) and how to use its two components, the Collector (Section 6.2) and the Analyzer (Section 6.3).

6.1 VAXELN Performance Utility Features

The Collector gathers performance data on a running application and writes the data to a performance data file. The Analyzer then reads the performance data file and produces performance tables displaying the data.

The Collector and the Analyzer are fully symbolic, obtaining symbol information from the debug symbol table that the compilers generate. The VAXELN Performance Utility collects and analyzes the following kinds of data:

- **Program Counter (PC) sampling data**. You specify which job to collect PC data for. The Collector will sample the program counter at an interval you select. The Analyzer can then produce reports telling you where your job is spending most of its time.
- **Job sampling data**. For each job in a system, the Collector returns the number of times the job entered the run state and the elapsed CPU time.

- **Process sampling data**. For each process in a specified job, the Collector returns the number of times the process entered the run state and the elapsed CPU time.
- **System service sampling data**. For a specified job, the Collector returns the number of times each of a subset of the VAXELN kernel services was called.

You can only select one type of data in a single Collector run, that is, each execution of a GO command terminated by a STOP command.

To use the VAXELN Performance Utility, the following minimum configuration is required:

- Two VAX computers connected by DECnet, one running VAX/VMS and the other running VAXELN. The Analyzer will run only on the VAX/VMS host.
- VAXELN Version 3.0 or higher.

NOTE

You cannot use the VAXELN Performance Utility for programs with a job priority of two or above and that call the INITIALIZATION_DONE procedure (**Init required** specified on the Program Description Menu).

The syntax for Collector and Analyzer commands is as follows:

COMMAND[[/qualifier . . .]] [[filespec]]

Qualifiers and parameters for Collector and Analyzer commands are described in Section 6.2.3 and Section 6.3.3, respectively.

6.2 Collecting Performance Data

To collect performance data from a program, you must prepare your VAXELN system for performance analysis, invoke the Collector, and issue Collector commands. Section 6.2.1 explains how to prepare your system for the Collector. Section 6.2.2 explains how to invoke the Collector. Section 6.2.3 explains the Collector commands.

6.2.1 Preparing for Performance Analysis

To prepare a VAXELN system for performance analysis, do the following:

1. For PC sampling, compile and link your program using the /DEBUG qualifier. The /DEBUG qualifier causes the compiler to create a debug symbol table. With the debug symbol table, the program image contains all the symbol and line number information needed for PC address and source code correlation. See Chapter 5 for a discussion of the debug symbol table. Although you compile and link with the /DEBUG qualifier, you do not have to select **Debug** on the System Builder's Editing System Characteristics and Editing Program Characteristics menus.

   For example, to compile MYFILE.PAS, type:

   ```
   $ EPASCAL/DEBUG/NOOP MYFILE + ELN$:RTLOBJECT/LIB
   ```

   To link your program, type:

   ```
   $ LINK/DEBUG MYFILE + ELN$:RTLSHARE/LIB + RTL/LIB
   ```

2. Build your program selecting EPA **Yes** at the System Builder's System Characteristics Menu (see Section 3.4.3). The System Builder includes the Collector in your system. Some of the pages you specify at the **System region size** entry on that menu are allocated to the Collector. The amount allocated depends on the kind of data you are collecting.

   - When you are doing job or process sampling, the number of **System region size** pages allocated is calculated as follows:

     ```
     ((50 * the number of jobs in the system) + 511)/512
     ```

   - When you are doing system service sampling, six **System region size** pages are allocated by the VAXELN Performance Utility.
   - When you are doing PC sampling, the number of **System region size** pages allocated is calculated as follows:

     ```
     ((4 * the code size in bytes/the bucket size) + 511)/512
     ```

     For a discussion of the bucket size refer to the SET PC_SAMPLING command in Section 6.2.3.

     NOTE

     Linking to the RTLOBJ library causes the Collector's datafile and the **System region size** to be much larger than does linking to the RTLSHARE library.

3. Load your system to the VAXELN target.

NOTE

When you include the Collector in your system, a job with a lower priority than the VAXELN Performance Utility, priority 3, will not start running until you initialize the Collector and issue the GO command (see Section 6.2.3).

6.2.2 Invoking the Collector

To invoke the Collector, enter the following command at the DCL prompt on the host system:

```
$ VAXELN PERFORMANCE/COLLECTOR target
```

where *target* is the DECnet name or number of the target node.

The following prompt appears:

```
EPC>
```

You can now use one of the Collector commands discussed in Section 6.2.3.

6.2.3 Collector Commands

Collector commands allow you to name the target and job for which data is collected, to define the kind of data collected, to specify the file that holds the data, to start and stop the Collector session on the target, and to exit the Collector at the host. The following example gives a set of commands for a complete Collector session.

```
EPC> SET NODE FRED
EPC> SET PC_SAMPLING/INTERVAL=20 MYFILE
EPC> SET DATA_FILE MYJOB.LOG
EPC> GO
EPC> STOP
EPC> EXIT
```

The first command specifies the system named FRED as the target for data collection. The second command specifies the kind of data to be collected as Program Counter samples for the program MYFILE and sets the sampling rate to 20 millisecond (ms) intervals. The third command makes MYJOB.LOG the file that will hold the data. The next two commands start and stop the Collector session on the target. The last command exits the Collector session at the host.

The rest of this section contains descriptions of the Collector commands in alphabetical order.

EXIT

Exits the Collector session.

Format

EXIT

Description

This command exits the Collector session at the host. If you use the EXIT command after a GO but before a STOP, the Collector session continues on the target. You can resume the Host's connection to the session by reissuing the command:

```
$ VAXELN PERFORMANCE/COLLECTOR target
```

GO

Tells the Collector to begin collecting the performance data.

Format

GO

Description

You must issue a SET DATAFILE command, as well as a command setting the kind of data to collect, before issuing a GO command. Before issuing a second GO command, you must terminate the session begun by the first with a STOP command.

HELP

Displays information on each of the Collector commands.

Format

HELP *⟦keyword . . . ⟧*

Parameter

keyword . . .
One or more keywords that refer to the topic or subtopic on which you want information. Information within HELP is arranged in a hierarchical manner. The levels are:

1. None — If you do not specify a keyword, HELP lists the topics that are documented in the help file. Each item in the list is a keyword in the first level of the hierarchy.
2. Topic — If you specify a keyword that names a topic, HELP describes the topic as it is documented in the help file. Keywords for additional information available on this topic are listed.
3. Topic subtopic — If you specify a subtopic following a topic, HELP provides a description of the specified subtopic.

Description

To use the HELP facility in its simplest form, issue the HELP command from your terminal. HELP displays a list of topics at your terminal. If you want to see more information on one of the topics, type HELP *topic* at the command level prompt. The system will display information on that topic. If the topic has subtopics, HELP will list the subtopics. If you want information on one of the subtopics, type HELP *topic subtopic* at the command level prompt.

Example

```
EPC> HELP GO
```

SET DATAFILE

Specifies the file the Collector uses to record performance data.

Format

SET DATAFILE *filespec*

Parameter

filespec
The file specification of the file to which the performance data is written.

Example

```
EPC> SET DATAFILE MYFILE.COL
```

SET JOB_SAMPLING

Enables the collection of entries to the run state and elapsed CPU times for a system's jobs.

Format

SET JOB_SAMPLING

Description

For each job in the system, this command returns the job identification number (refer to Section 5.3.1), the job name, the CPU time, and the number of times it entered the run state.

SET NODE

Changes the node from which you will collect performance data.

Format

SET NODE *target*

Parameter

target
A node name or number.

Example

```
EPC> SET NODE FRED
```

SET PC_SAMPLING

Enables the collection of program counter samples for the specified job, at the selected interval. PC data is registered if the process running when the sample is taken belongs to the specified job.

Format

SET PC_SAMPLING *job*

```
/BUCKET_SIZE=n
/CODE_START=n
/IMAGE_SIZE=n
/INTERVAL=n
```

Parameter

job
The program name entered on the Program Description Menu or specified in a LOAD command.

Qualifiers

/BUCKET_SIZE=n
Specifies the number of bytes each count in the tally represents. A bucket is a range of code. The Collector counts the number of times the executing command was in a particular range. The recommended bucket size is the minimum of 1. The maximum is 32767. The default is 16.

/CODE_START=n
Specifies the virtual address where sampling will begin. You would set this value if the job is loaded dynamically or if you do not wish to test the entire job processed by EBUILD. If you omit this qualifier for a job not processed by EBUILD, the Collector uses 200 (hexadecimal) as the starting address.

/IMAGE_SIZE=n
Specifies the size of the job being sampled. You would set this value if the job is loaded dynamically or if you do not wish to test the entire job processed by EBUILD. If you omit this qualifier for a job not processed by EBUILD, the Collector uses the system's P0 size as specified on the System Characteristics Menu.

/INTERVAL=n
Specifies the interval at which sampling occurs in milliseconds. The range is 1 through 32767. The default is the **Time interval** specified on the System Builder's System Characteristics Menu. If you enter a different value, it is truncated to the nearest multiple of the **Time interval.**

Example

```
EPC> SET PC_SAMPLING/BUCKET_SIZE=1 MYFILE
```

SET PROCESS_SAMPLING

Enables the collection of entries to the run state and elapsed CPU times for the processes in the specified job.

Format

SET PROCESS_SAMPLING *job*

Parameter

job
The name entered on the Program Description Menu. See Section 3.4.5.

Description

For each process in the job, this command returns the process identification number (refer to Section 5.3.1), the process name, if any, set by KER$NAME_OBJECT, the CPU time for the process, and the number of times it entered the run state.

Example

```
EPC> SET PROCESS_SAMPLING MYFILE
```

SET SYSTEM_SERVICE_SAMPLING

Returns the number of times each of the VAXELN kernel services listed in Table 6–1 is called by the specified job.

Format

SET SYSTEM_SERVICE_SAMPLING *job*

Parameter

job
The name entered on the Program Description Menu. See Section 3.4.5.

Description

This command keeps a count for each service called. The count, however, can be greater than the number of calls your job makes. Kernel routines can call other kernel routines. Each call to a routine is counted, whether the call comes from your job or from another kernel routine.

Table 6–1 lists the kernel service calls logged by this command.

Table 6–1: System Services Counted

	Service Name
	ACCEPT_CIRCUIT
	ALLOCATE_MAP
	ALLOCATE_MEMORY
	ALLOCATE_PATH
	ALLOCATE_SYSTEM_REGION
	CLEAR_EVENT
	CONNECT_CIRCUIT
REA	CREATE
	CREATE_DEVICE
	CREATE_EVENT

Table 6–1 (Cont.): System Services Counted

Service Name
CREATE_JOB
CREATE_MESSAGE
CREATE_NAME
CREATE_PORT
CREATE_PROCESS
CREATE_SEMAPHORE
CURRENT_PROCESS
DELETE
DISABLE_ASYNCH_EXCEPTION
DISABLE_SWITCH
DISCONNECT_CIRCUIT
ENABLE_ASYNCH_EXCEPTION
ENABLE_SWITCH
ENTER_KERNEL_CONTEXT
EXIT
FREE_MAP
FREE_MEMORY
FREE_PATH
FREE_SYSTEM_REGION
GET_JCB
GET_TIME
GET_USER
INITIALIZATION_DONE
INSTALL_PROGRAM
JOB_PORT
MEMORY_SIZE
POST_ERRORLOG
RAISE_DEBUG_EXCEPTION
RAISE_EXCEPTION

Table 6–1 (Cont.): System Services Counted

Service Name
RAISE_PROCESS_EXCEPTION
RECEIVE
REMOVE_PROGRAM
RESUME
SEND
SET_JOB_ELIGIBILITY
SET_JOB_PRIORITY
SET_PROCESS_PRIORITY
SET_PROTECTION
SET_TIME
SET_USER
SIGNAL
SIGNAL_DEVICE
SUSPEND
SYSTEM_SERVICES
TRANSLATE_NAME
UNWIND
WAIT_ALL
WAIT_ANY

SHOW DATAFILE

Displays the current data file specification. This file is set by the SET DATAFILE command.

Format **SHOW DATAFILE**

SHOW NODE

Displays the name of the node from which you are collecting data. This can be the node specified when you entered the Collector or the node specified by the SET NODE command.

Format

SHOW NODE

SHOW RUN

Displays the type of data being collected and, where applicable, the name of the job being sampled. For PC sampling, it also displays the sizes of the interval, bucket, and image.

Format

SHOW RUN

STOP

Ends data collection at the target and writes the performance data file; does not exit from the Collector at the host.

Format

STOP

6.3 Using the Analyzer

The Analyzer reads the data file the Collector builds and produces reports according to your specifications. You can use the Analyzer anytime after the Collector has produced a performance data file.

To generate Analyzer output, you must invoke the Analyzer, and issue Analyzer commands. Section 6.3.1 explains Analyzer output. Section 6.3.2 explains how to invoke the Collector. Section 6.2.3 explains the Collector commands.

6.3.1 Analyzer Output

The Analyzer can produce the following kinds of tables:

- Performance data next to the names of routines and/or modules
- Performance data next to source program text
- Run state entry counts and CPU times next to the names of jobs or processes
- Counts of kernel routine calls next to the routine names

You type the TABULATE command to produce the tables:

```
EPA> TABULATE
```

For PC sampling data, you control how the TABULATE command presents the data with the /ROUTINE, /MODULE, and /SOURCE qualifiers. For example, if you use the /MODULE qualifier, each row in a table represents one module. The numbers on each row tell you how often the executed command was in that module.

You can control the amount of PC Sampling data a TABULATE command displays by using the /MAXIMUM=*n*, /MINIMUM=*n*, and /⟦NO⟧ZERO) qualifiers. The /MAXIMUM=*n* and /MINIMUM=*n* qualifiers exclude the display of items whose percentages fall above or below the value you specify. The /ZERO qualifier includes the display of items whose count is zero.

In Figure 6–1, a table displays PC sampling data by routine. The first column gives the routine names. The second column gives the number of PC counts. The third column gives the percentage of PC counts. The command to generate this table would be:

```
EPA> TABULATE/ROUTINE/ZERO
```

Figure 6–1: PC Sampling Data by Routine

```
               VAXELN Performance Analyzer

             Program Counter Sampling Data

Module/Routine                       Count     Percent
MULT/                                 1592      100.0%
  A                                    160       10.1%
  B                                    291       18.3%
  C                                    629       39.5%
  MULT                                 512       32.2%
PAS$INPUT/                               0        0.0%
  $CODE                                  0        0.0%
PAS$OUTPUT/                              0        0.0%
  $CODE                                  0        0.0%
```

The information at the bottom of Figure 6–1 includes information displayed by all Analyzer tables and information only displayed for PC sampling data. The TABULATE command always reports its own beginning and ending times and displays the name of the Collector data file. For a description of the information that is specific to PC sampling data, see the discussion of the TABULATE command in Section 6.3.3.

The Collector commands to generate the data file for Figure 6–1 might be:

```
EPC> SET NODE TARGET_1
EPC> SET PC_SAMPLING MULT/BUCKET_SIZE=2
EPC> SET DATA_FILE MULT.PC
EPC> GO
EPC> STOP
EPC> EXIT
```

In Figure 6–2, PC sampling data for the same program run is displayed by source line. The first column gives the percentage of PC counts. The second column gives the number of PC counts. The third column gives the source line number followed by the source code text.

The command to generate this table would be:

```
EPA> TABULATE/SOURCE/ZERO
```

Figure 6–2: PC Sampling Data by Source Line

```
                              VAXELN Performance Analyzer

                             Program Counter Sampling Data

Percent     Count      Line
                         1:   PROGRAM mult ( input, output ) ;
                         2:
                         3:   PROCEDURE a;
                         4:
                         5:
                         6:       VAR
                         7:    i : integer;
                         8:    j : integer;
  0.0%          0        9:       BEGIN
  0.0%          0       10:       j := 0;
  5.4%         86       11:       FOR i := 1 TO 100 DO
  4.6%         74       12:    j := i + 1
                        13:       END;
                        14:
                        15:   PROCEDURE b;
                        16:
                        17:
                        18:       VAR
                        19:    i : integer;
                        20:    j : integer;
  0.0%          0       21:       BEGIN
  0.0%          0       22:       j := 0;
  8.7%        138       23:       FOR i := 1 TO 200 DO
  9.6%        153       24:    j := i + 1
                        25:       END;
                        26:
                        27:   PROCEDURE c;
                        28:
                        29:
                        30:       VAR
                        31:    i : integer;
                        32:    j : integer;
  0.0%          0       33:       BEGIN
  0.0%          0       34:       j := 0;
 19.2%        305       35:       FOR i := 1 TO 400 DO
 20.4%        324       36:    j := i + 1
                        37:       END;
                        38:
                        39:   VAR
                        40:       ii : integer;
                        41:       jj : integer;
                        42:
```

Figure 6–2 Cont'd. on next page

Figure 6–2 (Cont.): PC Sampling Data by Source Line

```
 0.0%         0     43:   BEGIN
                    44:   while true do
                    45:       begin
 0.0%         0     46:       jj := 0;
 4.3%        68     47:       for ii := 1 to 100 do
 5.9%        94     48:    jj := ii + 1;
 0.0%         0     49:       a;
 0.1%         1     50:       jj := 0;
 0.0%         0     51:       for ii := 1 to 100 do
11.9%       190     52:    jj := ii + 1;
 0.1%         2     53:       b;
 0.1%         1     54:       jj := 0;
 4.8%        77     55:       for ii := 1 to 100 do
 4.6%        74     56:    jj := ii + 1;
 0.3%         5     57:       c
                    58:       end
                    59:   END.


TABULATE Command Summary information:

Datafile = mult.pc

System file       : MULT
Begin time        : 17-NOV-1987 00:00:50.67
Stop time         : 17-NOV-1987 00:01:16.59
Clock rate        : 10 ms
Sampling interval : 10 ms

Total data points : 1592
Bucket size       : 2 bytes
Traced S0 counts  : 1
Untraced S0 counts: 1
```

In Figure 6–3, a table displays job sampling data. The table displays the data for each job in a system. The first column gives the job number assigned by the system when it creates the job. The second column gives the job name entered on the System Builder's Program Description Menu. The third column gives the number of times the job entered the run state. The fourth column gives the job's CPU time.

Figure 6–3: Job Sampling Data

```
                         VAXELN Performance Analyzer

                              JOB Sampling Data

  JOB      NAME                                  CONTEXT         TIME

    2     XQDRIVER                                  518        0.54 sec
    3     CONSOLE                                   934        5.60 sec
    4     EDEBUGREM                                   6        0.02 sec
    5     EPACEMAIN                                  15        0.05 sec
    6     FALSERVER                                   2        0.01 sec
    7     EDISPLAY                                  941        6.18 sec


TABULATE Command Summary information:

Datafile = EPACEMAIN.JOB

Begin time         : 17-JUL-1987 00:02:15.81
Stop time          : 17-JUL-1987 00:02:46.04
```

The Collector commands to generate the data file for Figure 6–3 might be:

```
EPC> SET NODE TARGET_1
EPC> SET JOB_SAMPLING
EPC> SET DATA_FILE EPACEMAIN.JOB
EPC> GO
EPC> STOP
EPC> EXIT
```

In Figure 6–4, a table displays process sampling data. The table displays the data for each process in job EPACEMAIN. The first column gives the process number assigned by the system when it creates the process. The second column gives the process name as specified in the program. The third column gives the number of times the process entered the run state. The fourth column gives the processes CPU time.

Figure 6–4: Process Sampling Data

```
                        VAXELN Performance Analyzer

                           PROCESS Sampling Data

JOB        NAME                                CONTEXT         TIME

   5     EPACEMAIN                                15         0.07 sec

 PROCESS(ES)

   1     EPACEMAIN                                 1         0.02 sec
   2     COLLECTOR_PROCESS                        14         0.05 sec


TABULATE Command Summary information:

Datafile = EPACEMAIN.PRO

System file       : EPACEMAIN
Begin time        : 17-JUL-1987 00:08:03.20
Stop time         : 17-JUL-1987 00:10:10:15
```

The Collector commands to generate the data file for Figure 6–4 might be:

```
EPC> SET NODE TARGET_1
EPC> SET PROCESS_SAMPLING EPACEMAIN
EPC> SET DATA_FILE EPACEMAIN.PRO
EPC> GO
EPC> STOP
EPC> EXIT
```

In Figure 6–5, a table displays system service counts. The first column gives the name of the kernel routine. The second column gives the number of times the routine was called during the job.

Figure 6–5: System Service Sampling Data

```
                              VAXELN Performance Analyzer

                             SYSTEM SERVICE Sampling Data

accept_circuit                                        1
create_message                                        8
delete                                                9
disconnect_circuit                                    1
receive                                               10
send                                                  8
signal                                                1
wait_any                                              10



TABULATE Command Summary information:

Datafile = EDEBUGREM_SYS.LOG

System file        : EDEBUGREM
Begin time         : 17-NOV-1987 00:02:04.68
Stop time          : 17-NOV-1987 00:02:43.35
```

The Collector commands to generate the data file for Figure 6–5 might be:

```
EPC> SET NODE TARGET_1
EPC> SET SYSTEM_SERVICE_SAMPLING EDEBUGREM
EPC> SET DATA_FILE EDEBUGREM_SYS.LOG
EPC> GO
EPC> STOP
EPC> EXIT
```

As the above example shows, system service count tables also display the name of the .SYS file.

Section 6.3.3 gives a complete discussion of the TABULATE command.

6.3.2 Invoking the Analyzer

To invoke the Analyzer, enter the following command at the DCL prompt on the host system:

```
$ VAXELN PERFORMANCE/ANALYZER  filespec
```

where *filespec* is the name of a performance data file created by the Collector. If you omit the file specification, the system prompts you for it.

The screen displays the Analyzer prompt:

```
EPA>
```

If the data file you specify contains PC sampling data, the Analyzer looks for a .EXE file with the same name in the same directory. If the Analyzer cannot find the .EXE file, the following prompt appears:

```
Enter image file name:
```

Enter the file specification of the program whose PC sampling data is collected in the data file.

You can now use one of the Analyzer commands discussed in Section 6.3.3.

6.3.3 Analyzer Commands

Analyzer commands allow you to create tables from Collector files, to print the tables, or to write them to text files. This section contains descriptions of the Analyzer commands in alphabetical order.

EXIT

Exits the Analyzer session.

Format **EXIT**

FILE

Writes the output from the most recent TABULATE command to a text file.

Format

FILE *filespec*

Parameter

filespec
The file specification of the text file to which the output is written.

Example

```
EPA> FILE MYTBL.LOG
```

HELP

Displays information on each of the Analyzer commands.

Format

HELP *[[keyword . . .]]*

Parameter

keyword . . .
One or more keywords that refer to the topic or subtopic on which you want information. Information within HELP is arranged in a hierarchical manner. The levels are:

1. None — If you do not specify a keyword, HELP lists the topics that are documented in the help file. Each item in the list is a keyword in the first level of the hierarchy.
2. Topic — If you specify a keyword that names a topic, HELP describes the topic as it is documented in the help file. Keywords for additional information available on this topic are listed.
3. Topic subtopic — If you specify a subtopic following a topic, HELP provides a description of the specified subtopic.

Description

To use the HELP facility in its simplest form, issue the HELP command from your terminal. HELP displays a list of topics at your terminal. If you want to see more information on one of the topics, type HELP *topic* at the command level prompt. The system will display information on that topic. If the topic has subtopics, HELP will list the subtopics. If you want information on one of the subtopics, type HELP *topic subtopic* at the command level prompt.

Example

```
EPA> HELP TABULATE
```

PRINT

Prints the output of the most recent TABULATE command. The output is sent to the queue assigned the logical name SYS$PRINT.

Format

PRINT

TABULATE

Generates tables from Collector data files. The qualifiers for this command control the display of PC sampling data and are ignored for tables displaying other kinds of data.

Format

TABULATE *⟦range . . . ⟧*

```
/MODULE
/ROUTINE
/SOURCE
/MAXIMUM=n
/MINIMUM=n
/⟦NO⟧ZERO
```

Parameter

range . . .
A list of one or more routine names or module names for which PC sampling data is to be displayed. If you omit this parameter for PC sampling data, the Analyzer returns data from the entire program address range.

Qualifiers

/MODULE
Specifies that the results are displayed by module.

/ROUTINE
Specifies that the results are displayed by routine; this is the default.

/SOURCE
Specifies that the source lines of the routines or modules identified by the *range* parameter are included in the display.

NOTE

The Collector counts the number of times machine instructions, not source instructions, are executed. Therefore, Analyzer tables may show counts where no corresponding source code exists.

/MAXIMUM=n
Specifies the maximum percentage to display.

/MINIMUM=n
Specifies the minimum percentage to display.

/[[NO]]ZERO
Specifies that routines or modules whose counts are zero will be included in the display; NOZERO is the default.

Description

The /ROUTINE, /MODULE, and /SOURCE qualifiers specify the partitioning of the table into rows. The numerical values on a row show the Collector's results for a routine, module, or source line. The /ROUTINE, /MODULE, and /SOURCE qualifiers are mutually exclusive.

In addition to performance data, the tables display the beginning and ending times of the TABULATE command's execution, the name of the Collector data file, and, for system service count tables, the name of the .SYS file. For PC sampling data, tables also display the following items:

- **System file**. The name of the image for which the table is displaying data
- **Clock rate**. The **Time interval** selected on the System Characteristics Menu
- **Sampling interval**. The value specified by the /INTERVAL qualifier to the Collector's SET PC_SAMPLING command
- **Total data points**. The total number of samples taken and traced to a P0 address in your code
- **Bucket size**. The value specified by the /BUCKET_SIZE qualifier to the Collector's SET PC_SAMPLING command
- **Traced S0 counts**. The number of times your program was found in S0 space and the address could be traced to a P0 address in your code

- **Untraced S0 counts**. The number of times your program was found in S0 space and the address could not be traced back to a P0 address in your code

Example

```
EPA> TABULATE/SOURCE/ZERO/MINIMUM=5
```

Appendix A

VAX-11/750 Microcode Patch

System Revision Level 5 of the VAX–11/750 and VAX–11/751 computers lets you patch the machine's microcode control store. The 11/750 runs without the patch, but not at the latest revision level.

You must load the microcode control store patches at system power-up. To do this in VAXELN, include a special program in each system you load onto the computer. The program must be compiled on site to take advantage of the latest patch set.

NOTE

> If your development system is not a VAX–11/750, you might have to copy the current patch file into the system's SYS$SYSTEM directory. Once built into a VAXELN system, the program continues running and reloads the microcode in the event of a power failure.

If in doubt about the revision level of your machine, check with your local DIGITAL field service representative.

A.1 Procedure

Perform the following procedures in the given order; they must be performed on a VAX-11/750 running VAX/VMS:

1. Set the default directory to ELN$.

   ```
   $ SET DEFAULT ELN$
   ```

2. Define the command to convert the patch file to an object file.

   ```
   $ SET COMMAND DATATOBJ
   ```

3. Create the patch's object file.

   ```
   $ DATATOBJ SYS$SYSTEM:PCS750.BINPCS750.OBJ
   ```

4. Link the resulting object file with the 11/750 microcode patch utility.

   ```
   $ LINK P750UCODE+PCS750.OBJ+RTLSHARE/LIBRARY+RTL/LIBRARY
   ```

5. Include P750UCODE.EXE with a System Builder program description, using the following characteristics:

Init required	**Yes**
Mode	**Kernel**
Job priority	**1**
Powerfailure exception	**Yes**

Appendix B
Using the Error Log Server

If you select VAXELN error logging for your system, you can create the error log file on the target or on a remote node (see Section 3.4.9). To create the file on a remote system you must run the Error Log Server (ELSE) on the remote node.

This appendix explains how to use the Error Log Server. Section B.1 gives an overview of ELSE. Section B.2 explains how to start and exit from ELSE. Section B.3 explains the ELSE parameters. And Section B.4 explains the qualifiers to the DCL RUN command that executes ELSE.

B.1 Overview of ELSE

ELSE must be running before the target attempts to send error logging information to the remote node. The impact of ELSE on the host VAX/VMS system is normally minimal, since ELSE remains inactive until it receives a message from the target. The minimum activity for a target node is to generate a time-stamp log once every 10 minutes.

The impact on the VAX/VMS system increases in direct relation to the fault activity on a target node, but does not increase greatly. The total impact is a function of both the number of nodes being serviced and the fault activity on those nodes. The impact on disk space is also a function of the number of nodes and the fault activity on each.

The error log files created by ELSE have names of the format *node*.SYS, where *node* is the name of the target node supplied in the DECnet connect message. ELSE also creates the files ELSE.LOG, for messages generated by ELSE, and ELSE.ERR, for messages generated by VAX/VMS. ELSE does not delete or purge these files automatically.

B.2 Invoking and Stopping ELSE

The following privileges are required to run ELSE:

Privilege	Meaning
SYSNAM	Allows inserting logical names in the system name table
NETMBX	Allows creating network connections
PRMMBX	Allows creating permanent mailboxes
DETACH	Allows creating a detached process

You invoke ELSE from the supplied procedure ELN$:ELSE$STARTUP.COM. The command to run the ELSE start-up procedure is:

```
$ @ELN$:ELSE$STARTUP
```

You can include this command in the system start-up procedure. You must start DECnet before you type this command. If you start DECnet by a batch job, you must issue the ELSE start-up command from the batch file.

You stop ELSE by running the supplied procedure ELN$:ELSE$-SHUTDOWN.COM. Do not attempt to stop ELSE by using the DCL STOP command. The command to run the ELSE shut-down procedure is:

```
$ @ELN$:ELSE$SHUTDOWN
```

You can include this command in the system shut-down procedure. If you issue this command before you shut down DECnet, ELSE.LOG indicates a user-requested termination. If DECnet is shut down first, ELSE will process any transactions in its queue and perform an automatic orderly termination, but ELSE.LOG will indicate a termination forced by a network shut down. Letting DECnet shut down first minimizes the possibility of missing any target messages.

NOTE

Whenever ELSE shuts down for any reason, you must restart it manually. It will not restart itself.

When a network link to a target node is lost, ELSE waits the time prescribed by the timer parameter ELSE$TIMER_DELAY (see Section B.3.4). If a target does not send a message in that time, ELSE assumes that the target has crashed and writes an inactivity message to the target's error log file.

B.3 ELSE Parameters

The VAX/VMS system logical names listed in Table B–1 act as run-time parameters for ELSE. The ELSE$STARTUP.COM procedure file assigns them their values. You can edit that file to change their values.

Table B–1: ELSE Parameters

Logical Name	Description
ELSE$ERRORLOG	The directory where the target error logs will be written
ELSE$LINK_IDLE_STATE	Determines whether the DECnet link will be maintained or dropped between messages from a target node
ELSE$SERVER_NAME	The name by which ELSE will be known on the network
ELSE$TIMER_DELAY	The length of time ELSE will wait before it assumes that a target node has crashed

B.3.1 ELSE$ERRORLOG

This parameter specifies the directory where the target error logs will be written. The default is the directory assigned to the system logical name SYS$ERRORLOG.

B.3.2 ELSE$LINK_IDLE_STATE

A value of UP specifies that the DECnet link will be maintained between messages from a target node. A value of DOWN specifies that the link will be dropped between messages. ELSE and each of the target nodes must be consistent with respect to this parameter. The default is DOWN.

B.3.3 ELSE$SERVER_NAME

This parameter specifies the name by which ELSE will be known on the network. ELSE and each of the target nodes must be consistent with respect to this parameter. The default is ELSE$SERVER.

B.3.4 ELSE$TIMER_DELAY

This parameter specifies a length of time in the format *hh:mm:sec*. If ELSE does not receive a message from a target node in that time, ELSE assumes the node has crashed. The timer is reset every time a message is received from a target. The default is 00:10:00.

B.4 RUN Command Qualifiers for ELSE

In addition to modifying ELSE parameters, you can modify the qualifiers to the RUN command in the ELSE$STARTUP.COM procedure. The qualifiers are set to their default values in ELSE$STARTUP.COM. Table B–2 lists those qualifiers and their defaults.

Table B–2: RUN Command Qualifiers for ELSE

Qualifier	Default	Comment
/AST_LIMIT=*quota*	100	Must be at least two plus number of target nodes.
/BUFFER_LIMIT=*quota*	80000	
/ENQUEUE_LIMIT=*quota*	200	
/ERROR=*filespec*	ELSE.ERR	The place where VAX/VMS-generated error messages will be sent.
/IO_BUFFERED=*quota*	100	
/IO_DIRECT=*quota*	50	
/OUTPUT=*filespec*	ELSE.LOG	The place where ELSE-generated messages will be sent.
/PRIORITY=*n*	7	
/PROCESS_NAME=*name*	ELSE$SERVER	The VAX/VMS process name, not the server's DECnet name.
/QUEUE_LIMIT=*quota*	75	
/UIC=*[uic]*	[1,6]	

Appendix C

A Full System Map

Figure C–1 is a full system map produced by the System Builder. To generate a full map, you specify the /MAP qualifiier to the EBUILD command together with the /FULL or /NOBRIEF qualifier. See Section 3.2.1.

The contents of a full map, in the following order, are:

- The .SYS file
- The VAXELN kernel corresponding to your choice on the Select Target Processor Menu, with internal data
- The VAXELN system files included in your system, with their defined characteristics and internal data
- The user files included in your system, with their defined characteristics and internal data
- The devices included in your system, with their defined characteristics
- The shareable images included in your system, with internal data
- The network node characteristics defined for your system
- The system characteristics defined for your system
- The size of your system in pages and bytes

Figure C–1:   A Full System Map

```
VAXELN System Builder                         13-JUL-1987 09:13:12.77
ELN T3.0-00

System file
-----------

TEST                    DISK$USER1:[MAIN.TMP]TEST.SYS;15

Kernel
------
ELN$:UV2KER.EXE         SYS$SYSDEVICE:[ELN]UV2KER.EXE;6

                        (VAXELN kernel)
                        Vectors and Data:   Start:  80000000  Pages:   5
                        Parameters:         Start:  80000A00  Bytes: 130
                        R/O Data and Code:  Start:  80000A84  Pages:  53
                        Transfer address:  00000000

Programs
--------
XQDRIVER                SYS$SYSDEVICE:[ELN]XQDRIVER.EXE;1

                        (Network device driver)
                        No debug, Run, Initialize, Mode = Kernel
                        User stack = 1, Kernel stack = 8
                        Job priority = 1, Process priority = 8
                        Job message limit = 16384
                        Power recovery exception = Disabled
                        Argument(s):
                            1)  "XQA"

                        Image section(s):
                          Type          Base VA    Page(s)   Image
                          Demand zero   00000200         2
                          Read-only     00000600        15
                          Fixup vector  00002400         1
                          Shareable     00002600        51   NETWORK
                          Fixup vector  00008C00         1
                        Transfer address:  00000644
```

Figure C–1 Cont'd. on next page

Figure C–1 (Cont.): A Full System Map

```
CONSOLE                  SYS$SYSDEVICE:[ELN]CONSOLE.EXE;1

                         (Terminal driver)
                         No debug, Run, Initialize, Mode = Kernel
                         User stack = 1, Kernel stack = 8
                         Job priority = 2, Process priority = 8
                         Job message limit = 16384
                         Power recovery exception = Disabled
                         Argument(s):
                             1)  "CONSOLE"

                         Image section(s):
                           Type          Base VA     Page(s)   Image
                           Noshr Write   00000200        1
                           Read-only     00000400        5
                           Fixup vector  00000E00        1
                         Transfer address:  00000428

EDEBUGREM                SYS$SYSDEVICE:[ELN]EDEBUGREM.EXE;1

                         (Remote debugger)
                         No debug, Run, Initialize, Mode = Kernel
                         User stack = 1, Kernel stack = 2
                         Job priority = 3, Process priority = 8
                         Job message limit = 16384
                         Power recovery exception = Disabled
                         Argument(s):

                         Image section(s):
                           Type          Base VA     Page(s)   Image
                           Demand zero   00000200        1
                           Read-only     00000400       21
                           Fixup vector  00002E00        1
                         Transfer address:  000019DD

FALSERVER                SYS$SYSDEVICE:[ELN]FALSERVER.EXE;1

                         (File access listener)
                         No debug, Run, Initialize, Mode = User
                         User stack = 1, Kernel stack = 1
                         Job priority = 16, Process priority = 8
                         Job message limit = 16384
                         Power recovery exception = Disabled
                         Argument(s):

                         Image section(s):
                           Type          Base VA     Page(s)   Image
                           Read-only     00000200       11
                           Fixup vector  00001800        1
                         Transfer address:  0000020F
```

Figure C–1 Cont'd. on next page

Figure C–1 (Cont.): A Full System Map

```
TEST                    DISK$USER1:[MAIN.TMP]TEST.EXE;15

                        Debug, Run, No initialize, Mode = User
                        User stack = 1, Kernel stack = 4
                        Job priority = 16, Process priority = 8
                        Job message limit = 16384
                        Power recovery exception = Disabled
                        Argument(s):

                        Image section(s):
                          Type          Base VA    Page(s)  Image
                          Noshr Write   00000200       1
                          Demand zero   00000400     195
                          Read-only     00018A00       1
                          Fixup vector  00018C00       2
                          Shareable     00019000      18    CMSC
                          Fixup vector  0001B400       1
                          Read-only     0001B600      38    DCIO (1)
                          Noshr Write   00020200       1    DCIO (2)
                          Fixup vector  00020400       2
                        Transfer address:  00018A00

Devices
-------
CONSOLE                 CSR address = %00000000
                        Vector = %0370
                        Priority = 4
                        BI number = 0
                        Adapter number = 0

XQA                     CSR address = %0774440
                        Vector = %0120
                        Priority = 4
                        BI number = 0
                        Adapter number = 0

Terminals
---------
CONSOLE                 Hardcopy, Escape, Echo, No pass all, No eight-bit
```

Figure C–1 Cont'd. on next page

Figure C-1 (Cont.): A Full System Map

```
Shareable images
----------------
NETWORK                 SYS$SYSDEVICE:[ELN]NETWORK.EXE;1
                        Major Id: 1, Minor Id: 0
                        Map into program region = Yes
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     8000FA00        49
                          Noshr Write   80015C00         2
                          Fixup vector  80016000         1

PASCALMSC               SYS$SYSDEVICE:[ELN]PASCALMSC.EXE;1
                        Major Id: 1, Minor Id: 3
                        Map into program region = No
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80016400        10

TERMINAL                SYS$SYSDEVICE:[ELN]TERMINAL.EXE;1
                        Major Id: 1, Minor Id: 0
                        Map into program region = No
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80017800         9
                          Fixup vector  80018A00         1

DAP                     SYS$SYSDEVICE:[ELN]DAP.EXE;1
                        Major Id: 1, Minor Id: 1
                        Map into program region = No
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80018C00        53
                          Fixup vector  8001F600         1

PRGLOADER               SYS$SYSDEVICE:[ELN]PRGLOADER.EXE;1
                        Major Id: 1, Minor Id: 0
                        Map into program region = No
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     8001F800         8
                          Fixup vector  80020800         1
```

Figure C-1 Cont'd. on next page

Figure C–1 (Cont.): A Full System Map

```
CMSC                    SYS$SYSDEVICE:[ELN]CMSC.EXE;1
                        Major Id: 2, Minor Id: 0
                        Map into program region = Yes
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80020A00        17
                          Noshr Write   80022C00         1
                          Fixup vector  80022E00         1

DCIO                    SYS$SYSDEVICE:[ELN]DCIO.EXE;1
                        Major Id: 2, Minor Id: 0
                        Map into program region = Yes
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80023200        38
                          Noshr Write   80027E00         1
                          Fixup vector  80028000         2

ELNACCESS               SYS$SYSDEVICE:[ELN]ELNACCESS.EXE;1
                        Major Id: 1, Minor Id: 0
                        Map into program region = No
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80028A00         4
                          Fixup vector  80029200         1

VAXEMUL                 SYS$SYSDEVICE:[ELN]VAXEMUL.EXE;1
                        Major Id: 1, Minor Id: 0
                        Map into program region = No
                        Image section(s):
                          Type          Base VA    Page(s)
                          Read-only     80029400        18

Network node characteristics
----------------------------
Network service         Yes
Name server             Yes
File access listener    Yes
Network device          QNA
Node name
Node address            0
Authorization required  No
Authorization service   None
Authorization file      AUTHORIZE.DAT
Default system UIC      [1,1]
Node triggerable        Yes
Network segment size    576 bytes
Remote command language No
```

Figure C–1 Cont'd. on next page

Figure C–1 (Cont.): A Full System Map

```
System characteristics
----------------------
Target processor          MicroVAX II (KA630)
Debugger                  Remote
Console driver            Yes
Instruction emulation     String
Boot method               Downline-load
Volume/device names
Guaranteed image list
Number of Jobs            16
Number of Subprocesses    48
Ports                     256
Pool size                 384 blocks
P0 Virtual size           1024 pages
P1 Virtual size           128 pages
Interrupt stack           2 pages
System region size        128 pages
Dynamic program space     0 pages
Time interval             10000 microseconds
Connect time              45 seconds
Memory limit              0 pages
Error logging             No
EPA                       No

System image size is 348 pages (174K bytes)

/NOEDIT/MAP/FULL TEST
```

Index

HOW TO ORDER

ADDITIONAL DOCUMENTATION

From	Call	Write
Alaska, Hawaii, or New Hampshire	603–884–6660	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Rest of U.S.A. and Puerto Rico*	800–258–1710	
* Prepaid orders from Puerto Rico must be placed with DIGITAL's local subsidiary (809–754–7575)		
Canada	800–267–6219 (for software documentation) 613–592–5111 (for hardware documentation)	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: Direct Order desk
Internal orders (for software documentation)	—	Software Distribution Center (SDC) Digital Equipment Corporation Westminster, MA 01473
Internal orders (for hardware documentation)	617–234–4323	Publishing & Circulation Serv. (P&CS) NR03–1/W3 Digital Equipment Corporation Northboro, MA 01532

Reader's Comments

VAXELN Host System Guide
AA–JG87B–TE

Your comments and suggestions will help us improve the quality of our future documentation. Please note that this form is for comments on documentation only.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (product works as described)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What I like best about this manual: ______________________________

____

What I like least about this manual: ______________________________

____

My additional comments or suggestions for improving this manual:

____

____

I found the following errors in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________

Please indicate the type of user/reader that you most nearly represent:

☐ Administrative Support	☐ Scientist/Engineer
☐ Computer Operator	☐ Software Support
☐ Educator/Trainer	☐ System Manager
☐ Programmer/Analyst	☐ Other (please specify) ______________
☐ Sales	

Name/Title ______________________________ Dept. ______________

Company ______________________________ Date ______________

Mailing Address ______________________________

______________________________ Phone ______________

10/87

— — Do Not Tear — Fold Here and Tape — — — — — — — — — — — — — — — — — — —

digital™

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

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

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
CORPORATE USER PUBLICATIONS
MLO5–5/E45
146 MAIN STREET
MAYNARD, MA 01754–2571

— — Do Not Tear — Fold Here —